

# C++ AMP : Language and Programming Model

## Version 1.2, December 2013

© 2013 Microsoft Corporation. All rights reserved.

This specification reflects input from PathScale Inc, NVIDIA Corporation (Nvidia) and Advanced Micro Devices, Inc. (AMD).

**Copyright License.** Microsoft grants you a license under its copyrights in the specification to (a) make copies of the specification to develop your implementation of the specification, and (b) distribute portions of the specification in your implementation or your documentation of your implementation.

**Patent Notice.** Microsoft provides you certain patent rights for implementations of this specification under the terms of Microsoft's Community Promise, available at <http://www.microsoft.com/openspecifications/en/us/programs/community-promise/default.aspx>.

**THIS SPECIFICATION IS PROVIDED "AS IS." MICROSOFT MAY CHANGE THIS SPECIFICATION OR ITS OWN IMPLEMENTATIONS AT ANY TIME AND WITHOUT NOTICE. MICROSOFT MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, (1) AS TO THE INFORMATION IN THIS SPECIFICATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; OR (2) THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS OR OTHER RIGHTS.**

## **ABSTRACT**

C++ AMP (Accelerated Massive Parallelism) is a native programming model that contains elements that span the C++ programming language and its runtime library. It provides an easy way to write programs that compile and execute on data-parallel hardware, such as graphics cards (GPUs).

The syntactic changes introduced by C++ AMP are minimal, but additional restrictions are enforced to reflect the limitations of data parallel hardware.

Data parallel algorithms are supported by the introduction of multi-dimensional array types, array operations on those types, indexing, asynchronous memory transfer, shared memory, synchronization and tiling/partitioning techniques.

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Overview .....</b>  | <b>1</b> |
| 1.1      | Conformance .....  | 1        |
| 1.2      | Definitions.....   | 2        |
| 1.3      | Error Model.....   | 4        |
| 1.4      | Programming Model .....  | 5        |
| <b>2</b> | <b>C++ Language Extensions for Accelerated Computing .....</b> | <b>6</b> |
| 2.1      | Syntax.....  | 6        |
| 2.1.1    | Function Declarator Syntax.....                                | 7        |
| 2.1.2    | Lambda Expression Syntax.....                                  | 7        |
| 2.1.3    | Type Specifiers .....  | 8        |
| 2.2      | Meaning of Restriction Specifiers .....                        | 8        |
| 2.2.1    | Function Definitions.....                                      | 9        |
| 2.2.2    | Constructors and Destructors .....                             | 9        |
| 2.3      | Expressions Involving Restricted Functions .....               | 10       |
| 2.3.1    | Function pointer conversions .....                             | 10       |
| 2.3.2    | Function Overloading.....                                      | 10       |
| 2.3.2.1  | Overload Resolution.....                                       | 11       |
| 2.3.2.2  | Name Hiding.....   | 12       |
| 2.3.3    | Casting.....   | 12       |
| 2.4      | amp Restriction Modifier .....                                 | 13       |
| 2.4.1    | Restrictions on Types .....                                    | 13       |
| 2.4.1.1  | Type Qualifiers .....  | 13       |
| 2.4.1.2  | Fundamental Types .....  | 13       |
| 2.4.1.3  | Compound Types .....   | 14       |
| 2.4.2    | Restrictions on Function Declarators.....                      | 14       |
| 2.4.3    | Restrictions on Function Scopes .....                          | 14       |
| 2.4.3.1  | Literals.....  | 14       |

|           |  |           |
|-----------|--|-----------|
| 2.4.3.2   | Primary Expressions (C++11 5.1) .....      | 14        |
| 2.4.3.3   | Lambda Expressions .....                   | 15        |
| 2.4.3.4   | Function Calls (C++11 5.2.2) .....         | 15        |
| 2.4.3.5   | Local Declarations .....                   | 15        |
| 2.4.3.5.1 | tile_static Variables.....                 | 15        |
| 2.4.3.6   | Type-Casting Restrictions.....             | 15        |
| 2.4.3.7   | Miscellaneous Restrictions.....            | 16        |
| <b>3</b>  | <b>Device Modeling .....</b>               | <b>16</b> |
| 3.1       | The concept of a compute accelerator ..... | 16        |
| 3.2       | accelerator .....                          | 16        |
| 3.2.1     | Default Accelerator .....                  | 16        |
| 3.2.2     | Synopsis .....                             | 17        |
| 3.2.3     | Static Members .....                       | 18        |
| 3.2.4     | Constructors.....                          | 19        |
| 3.2.5     | Members.....                               | 20        |
| 3.2.6     | Properties.....                            | 21        |
| 3.3       | accelerator_view.....                      | 22        |
| 3.3.1     | Synopsis .....                             | 23        |
| 3.3.2     | Queuing Mode .....                         | 23        |
| 3.3.3     | Constructors.....                          | 24        |
| 3.3.4     | Members.....                               | 24        |
| 3.4       | Device enumeration and selection API ..... | 26        |
| <b>4</b>  | <b>Basic Data Elements.....</b>            | <b>26</b> |
| 4.1       | index<N> .....                             | 26        |
| 4.1.1     | Synopsis .....                             | 27        |
| 4.1.2     | Constructors.....                          | 28        |
| 4.1.3     | Members.....                               | 29        |
| 4.1.4     | Operators.....                             | 29        |
| 4.2       | extent<N> .....                            | 31        |
| 4.2.1     | Synopsis .....                             | 31        |
| 4.2.2     | Constructors.....                          | 32        |
| 4.2.3     | Members.....                               | 33        |
| 4.2.4     | Operators.....                             | 34        |
| 4.3       | tiled_extent<D0,D1,D2> .....               | 35        |
| 4.3.1     | Synopsis .....                             | 36        |
| 4.3.2     | Constructors.....                          | 37        |
| 4.3.3     | Members.....                               | 38        |
| 4.3.4     | Operators.....                             | 38        |
| 4.4       | tiled_index<D0,D1,D2> .....                | 39        |
| 4.4.1     | Synopsis .....                             | 40        |

|          |  |           |
|----------|--|-----------|
| 4.4.2    | Constructors.....  | 42        |
| 4.4.3    | Members.....   | 42        |
| 4.5      | tile_barrier .....   | 43        |
| 4.5.1    | Synopsis .....   | 43        |
| 4.5.2    | Constructors.....  | 43        |
| 4.5.3    | Members.....   | 44        |
| 4.5.4    | Other Memory Fences .....                                      | 44        |
| 4.6      | completion_future.....   | 44        |
| 4.6.1    | Synopsis .....   | 45        |
| 4.6.2    | Constructors.....  | 45        |
| 4.6.3    | Members.....   | 46        |
| 4.7      | Access type .....  | 47        |
| <b>5</b> | <b>Data Containers .....</b>                                   | <b>47</b> |
| 5.1      | array<T,N> .....   | 47        |
| 5.1.1    | Synopsis .....   | 48        |
| 5.1.2    | Constructors.....  | 56        |
| 5.1.2.1  | Staging Array Constructors.....                                | 59        |
| 5.1.3    | Members.....   | 61        |
| 5.1.4    | Indexing.....  | 62        |
| 5.1.5    | View Operations .....  | 63        |
| 5.2      | array_view<T,N>.....   | 64        |
| 5.2.1    | Synopsis .....   | 66        |
| 5.2.1.1  | array_view<T,N> .....  | 66        |
| 5.2.1.2  | array_view<const T,N>.....                                     | 70        |
| 5.2.2    | Constructors.....  | 74        |
| 5.2.3    | Members.....   | 76        |
| 5.2.4    | Indexing.....  | 78        |
| 5.2.5    | View Operations .....  | 79        |
| 5.3      | Copying Data .....   | 81        |
| 5.3.1    | Synopsis .....   | 81        |
| 5.3.2    | Copying between array and array_view .....                     | 82        |
| 5.3.3    | Copying from standard containers to arrays or array_views..... | 84        |
| 5.3.4    | Copying from arrays or array_views to standard containers..... | 85        |
| <b>6</b> | <b>Atomic Operations.....</b>                                  | <b>85</b> |
| 6.1      | Synopsis .....   | 85        |
| 6.2      | Atomically Exchanging Values.....                              | 86        |
| 6.3      | Atomically Applying an Integer Numerical Operation .....       | 87        |
| <b>7</b> | <b>Launching Computations: parallel_for_each .....</b>         | <b>88</b> |
| 7.1      | Capturing Data in the Kernel Function Object .....             | 91        |
| 7.2      | Exception Behaviour .....                                      | 91        |

|           |   |            |
|-----------|---|------------|
| <b>8</b>  | <b>Correctly Synchronized C++ AMP Programs .....</b>                                | <b>91</b>  |
| 8.1       | Concurrency of sibling threads launched by a parallel_for_each call.....            | 91         |
| 8.1.1     | Correct usage of tile barriers .....  | 92         |
| 8.1.2     | Establishing order between operations of concurrent parallel_for_each threads ..... | 94         |
| 8.1.2.1   | Barrier-incorrect programs .....  | 94         |
| 8.1.2.2   | Compatible memory operations .....  | 94         |
| 8.1.2.3   | Concurrent memory operations.....   | 95         |
| 8.1.2.4   | Racy programs.....  | 96         |
| 8.1.2.5   | Race-free programs.....   | 96         |
| 8.2       | Cumulative effects of a parallel_for_each call .....                                | 96         |
| 8.3       | Effects of copy and copy_async operations.....                                      | 98         |
| 8.4       | Effects of array_view::synchronize, synchronize_async and refresh functions.....    | 99         |
| <b>9</b>  | <b>Math Functions .....</b>   | <b>100</b> |
| 9.1       | fast_math.....  | 100        |
| 9.2       | precise_math .....  | 103        |
| 9.3       | Miscellaneous Math Functions (Optional).....  | 110        |
| <b>10</b> | <b>Graphics (Optional) .....</b>  | <b>113</b> |
| 10.1      | texture<T,N>.....   | 113        |
| 10.1.1    | Synopsis .....  | 114        |
| 10.1.2    | Introduced typedefs .....   | 117        |
| 10.1.3    | Constructing an uninitialized texture .....   | 118        |
| 10.1.4    | Constructing a staging texture .....  | 119        |
| 10.1.5    | Constructing a texture from a host side iterator .....                              | 121        |
| 10.1.6    | Constructing a texture from a host-side data source.....                            | 122        |
| 10.1.7    | Constructing a texture by cloning another.....                                      | 123        |
| 10.1.8    | Assignment operator .....   | 124        |
| 10.1.9    | Copying textures.....   | 124        |
| 10.1.10   | Moving textures.....  | 125        |
| 10.1.11   | Querying texture's physical characteristics.....                                    | 125        |
| 10.1.12   | Querying texture's logical dimensions .....   | 125        |
| 10.1.13   | Querying the accelerator_view where the texture resides .....                       | 126        |
| 10.1.14   | Querying a staging texture's row and depth pitch .....                              | 126        |
| 10.1.15   | Reading and writing textures .....  | 126        |
| 10.1.16   | Direct3d Interop Functions.....   | 127        |
| 10.2      | writeonly_texture_view<T,N> .....   | 128        |
| 10.2.1    | Synopsis .....  | 128        |
| 10.2.2    | Introduced typedefs .....   | 128        |
| 10.2.3    | Construct a writeonly view over a texture .....                                     | 128        |
| 10.2.4    | Copy constructors and assignment operators.....                                     | 129        |
| 10.2.5    | Querying underlying texture's physical characteristics .....                        | 129        |

|          |   |     |
|----------|---|-----|
| 10.2.6   | Querying the underlying texture's accelerator_view .....                | 129 |
| 10.2.7   | Querying underlying texture's logical dimensions (through a view) ..... | 129 |
| 10.2.8   | Writing a write-only texture view.....                                  | 129 |
| 10.2.9   | Direct3d Interop Functions.....   | 130 |
| 10.3     | sampler .....   | 130 |
| 10.3.1   | Synopsis .....  | 130 |
| 10.3.2   | filter_modes .....  | 131 |
| 10.3.3   | address_mode .....  | 131 |
| 10.3.4   | Constructors .....  | 131 |
| 10.3.5   | Members .....   | 132 |
| 10.3.6   | Direct3d Interop Functions.....   | 133 |
| 10.4     | texture_view<T,N>.....  | 133 |
| 10.4.1   | Synopsis .....  | 133 |
| 10.4.2   | Introduced typedefs .....   | 134 |
| 10.4.3   | Constructors .....  | 134 |
| 10.4.4   | Copy constructors and assignment operators.....                         | 135 |
| 10.4.5   | Query functions .....   | 135 |
| 10.4.5.1 | Querying texture's physical characteristics.....                        | 135 |
| 10.4.5.2 | Querying texture's logical dimensions .....                             | 135 |
| 10.4.5.3 | Querying the accelerator_view where the texture resides .....           | 136 |
| 10.4.6   | Reading and writing a texture_view .....                                | 136 |
| 10.4.7   | Direct3d Interop Functions.....   | 137 |
| 10.5     | texture_view<const T,N>.....  | 137 |
| 10.5.1   | Synopsis .....  | 137 |
| 10.5.2   | Introduced typedefs .....   | 138 |
| 10.5.3   | Constructors .....  | 139 |
| 10.5.4   | Copy constructors and assignment operators.....                         | 140 |
| 10.5.5   | Query functions .....   | 140 |
| 10.5.5.1 | Querying texture's physical characteristics.....                        | 140 |
| 10.5.5.2 | Querying texture's logical dimensions .....                             | 140 |
| 10.5.5.3 | Querying the accelerator_view where the texture resides .....           | 141 |
| 10.5.6   | Indexing operations .....   | 141 |
| 10.5.7   | Sampling operations .....   | 141 |
| 10.5.8   | Gathering operations.....   | 142 |
| 10.6     | Global texture copy functions.....                                      | 144 |
| 10.6.1   | Global async texture copy functions .....                               | 147 |
| 10.7     | norm and unorm.....   | 147 |
| 10.7.1   | Synopsis .....  | 147 |
| 10.7.2   | Constructors and Assignment.....  | 149 |

|           |  |            |
|-----------|--|------------|
| 10.7.3    | Operators.....   | 149        |
| 10.8      | Short Vector Types.....                                      | 149        |
| 10.8.1    | Synopsis .....   | 150        |
| 10.8.2    | Constructors .....   | 151        |
| 10.8.2.1  | Constructors from components .....                           | 152        |
| 10.8.2.2  | Explicit conversion constructors .....                       | 152        |
| 10.8.3    | Component Access (Swizzling) .....                           | 152        |
| 10.8.3.1  | Single-component access.....                                 | 153        |
| 10.8.3.2  | Reference to single-component access.....                    | 153        |
| 10.8.3.3  | Two-component access.....                                    | 153        |
| 10.8.3.4  | Three-component access .....                                 | 154        |
| 10.8.3.5  | Four-component access .....                                  | 154        |
| 10.9      | Template Versions of Short Vector Types.....                 | 154        |
| 10.9.1    | Synopsis .....   | 155        |
| 10.9.2    | <code>short_vector&lt;T,N&gt;</code> type equivalences ..... | 157        |
| 10.10     | Template class <code>short_vector_traits</code> .....        | 158        |
| 10.10.1   | Synopsis .....   | 158        |
| 10.10.2   | Typedefs .....   | 161        |
| 10.10.3   | Members .....  | 162        |
| <b>11</b> | <b>D3D interoperability (Optional)</b> .....                 | <b>163</b> |
| 11.1      | <code>scoped_d3d_access_lock</code> .....                    | 166        |
| 11.1.1    | Synopsis .....   | 166        |
| 11.1.2    | Constructors .....   | 167        |
| 11.1.3    | Move constructors and assignment operators.....              | 167        |
| 11.1.4    | Destructor.....  | 167        |
| <b>12</b> | <b>Error Handling</b> .....                                  | <b>167</b> |
| 12.1      | <code>static_assert</code> .....                             | 167        |
| 12.2      | Runtime errors.....  | 168        |
| 12.2.1    | <code>runtime_exception</code> .....                         | 168        |
| 12.2.1.1  | Synopsis.....  | 168        |
| 12.2.1.2  | Constructors.....  | 168        |
| 12.2.1.3  | Members .....  | 169        |
| 12.2.1.4  | Specific Runtime Exceptions .....                            | 169        |
| 12.2.2    | <code>out_of_memory</code> .....                             | 169        |
| 12.2.2.1  | Synopsis.....  | 169        |
| 12.2.2.2  | Constructor .....  | 169        |
| 12.2.3    | <code>invalid_compute_domain</code> .....                    | 170        |
| 12.2.3.1  | Synopsis.....  | 170        |
| 12.2.3.2  | Constructor .....  | 170        |

|           |  |            |
|-----------|--|------------|
| 12.2.4    | unsupported_feature .....  | 170        |
| 12.2.4.1  | Synopsis.....  | 170        |
| 12.2.4.2  | Constructor .....  | 171        |
| 12.2.5    | accelerator_view_removed.....  | 171        |
| 12.2.5.1  | Synopsis.....  | 171        |
| 12.2.5.2  | Constructor .....  | 171        |
| 12.2.5.3  | Members .....  | 171        |
| 12.3      | Error handling in device code (amp-restricted functions) (Optional)..... | 172        |
| <b>13</b> | <b>Appendix: C++ AMP Future Directions (Informative).....</b>            | <b>173</b> |
| 13.1      | Versioning Restrictions .....  | 173        |
| 13.1.1    | <i>auto</i> restriction .....  | 173        |
| 13.1.2    | Automatic restriction deduction .....                                    | 174        |
| 13.1.3    | <i>amp</i> Version.....  | 174        |
| 13.2      | Projected Evolution of <i>amp</i> -Restricted Code.....                  | 175        |

## 1 Overview

C++ AMP is a compiler and programming model extension to C++ that enables the acceleration of C++ code on data-parallel hardware.

One example of data-parallel hardware today is the discrete graphics card (GPU), which is becoming increasingly relevant for general purpose parallel computations, in addition to its main function as a graphics accelerator. Another example of data-parallel hardware is the SIMD vector instruction set, and associated registers, found in all modern processors.

For the remainder of this specification, we shall refer to the data-parallel hardware as the *accelerator*. In the few places where the distinction matters, we shall refer to a GPU or a VectorCPU.

The programming model contains multiple layers, allowing developers to trade off ease-of-use with maximum performance. The data parallel computations performed on the accelerator are expressed using high-level abstractions, such as multi-dimensional arrays, high level array manipulation functions, and multi-dimensional indexing operations, all based on a large subset of the C++ programming language. The developer may use high level abstraction like `array_view` and delegate low level resource management to the runtime. Or the developer may explicitly manage all communication between the CPU and the accelerator, and this communication can be either synchronous or asynchronous.

C++ AMP is composed of three broad categories of functionality:

1. C++ language and compiler
  - a. Kernel functions are compiled into code that is specific to the accelerator.
2. Runtime
  - a. The runtime contains a C++ AMP abstraction of lower-level accelerator APIs, as well as support for multiple host threads and processors, and multiple accelerators.
  - b. Asynchronous execution is supported through an eventing model.
3. Programming model
  - a. A set of classes describing the shape and extent of data.
  - b. A set of classes that contain or refer to data used in computations
  - c. A set of functions for copying data to and from accelerators
  - d. A math library
  - e. An atomic library
  - f. A set of miscellaneous intrinsic functions

### 1.1 Conformance

*All text in this specification falls into one of the following categories:*

- **Informative:** shown in this style.

Informative text is non-normative; for background information only; not required to be implemented in order to conform to this specification.

- **Microsoft-specific:** shown in this style.

Microsoft-specific text is non-normative; for background information only; not required to be implemented in order to conform to this specification; explains features that are specific to the Microsoft implementation of the C++ AMP programming model. However, implementers are free to implement these feature, or any subset thereof.

- Normative: all text, unless otherwise marked (see previous categories) is normative. Normative text falls into the following two sub-categories:

- 47           ○ Optional: each section of the specification that falls into this sub-category includes the suffix “(Optional)”  
 48            in its title. A conforming implementation of C++ AMP may choose to support such features, or not.  
 49            (Microsoft-specific portions of the text are also Optional.)  
 50           ○ Required: unless otherwise stated, all Normative text falls into the sub-category of Required. A  
 51            conforming implementation of C++ AMP *must support all* Required features.

52       Conforming implementations shall provide all normative features and any number of optional features. Implementations  
 53       may provide additional features so long as these features are exposed in namespaces other than those listed in this  
 54       specification. Implementation may provide additional language support for amp-restricted functions (section 2.1) by  
 55       following the rules set forth in section 13.

56       The programming model utilizes *properties*. Any such property is optional. An implementation is free to use mechanisms  
 57       equivalent to Microsoft’s Visual C++ properties as long as they provide the same functionality of indirection to a member  
 58       function.

## 60       1.2 Definitions

61       This section introduces terms used within the body of this specification.

62       • **Accelerator**

63       A hardware device or capability that enables accelerated computation on data-parallel workloads. Examples  
 64       include:

- 65           ○ Graphics Processing Unit, or GPU, other coprocessor, accessible through the PCIe bus.
- 66           ○ Graphics Processing Unit, or GPU, or other coprocessor that is integrated with a CPU on the same die.
- 67           ○ SIMD units of the host node exposed through software emulation of a hardware accelerator.

68       • **Array**

69       A dense N-dimensional data container.

70       • **Array View**

71       A view into a contiguous piece of memory that adds array-like dimensionality.

72       • **Compressed texture format.**

73       A format that divides a texture into blocks that allow the texture to be reduced in size by a fixed ratio; typically 4:1  
 74       or 6:1. Compressed textures are useful when perfect image/texel fidelity is not necessary but where minimizing  
 75       memory storage and bandwidth are critical to application performance.

76       • **Extent**

77       A vector of integers that describes lengths of N-dimensional array-like objects.

78       • **Global memory**

79       On a GPU, global memory is the main off-chip memory store.

80       ***Informative:*** Typically, on current-generation GPUs, global memory is implemented in DRAM, with access times of  
 81       400-1000 cycles; the GPU clock speed is around 1 Ghz; and may or may not be cached. Global memory is accessed  
 82       in a coalesced pattern with a granularity of 128 bytes, so when accessing 4 bytes of global memory, 32 successive  
 83       threads need to read the 32 successive 4-byte addresses, to be fully coalesced.

84       The memory space of current GPUs is typically disjoint from its host system.

- 95     • **GPGPU:** General Purpose computation on Graphics Processing Units, which is a GPU capable of running non-  
96       graphics computations.
- 97     • **GPU:** A specialized (co)processor that offloads graphics computation and rendering from the host. As GPUs have  
98       evolved, they have become increasingly able to offload non-graphics computations as well (see GPGPU).
- 100
- 101     • **Heterogenous programming**  
102       A workload that combines kernels executing on data-parallel compute nodes with algorithms running on CPUs.
- 103
- 104     • **Host**  
105       The operating system process and the CPU(s) that it is running on.
- 106
- 107     • **Host thread**  
108       The operating system thread and the CPU(s) that it is running on. A host thread may initiate a copy operation or  
109       parallel loop operation that may run on an accelerator.
- 110
- 111     • **Index**  
112       A vector of integers that describes an N-dimentional point in iteration space or index space.
- 113
- 114     • **Kernel; Kernel function**  
115       A program designed to be executed at a C++ AMP call-site. More generally, a kernel is a unit of computation that  
116       executes on an accelerator. A kernel function is a special case; it is the root of a logical call graph of functions that  
117       execute on an accelerator. A C++ analogy is that it is the “[main\(\)](#)” function for an accelerator program
- 118
- 119     • **Perfect loop nest**  
120       A loop nest in which the body of each outer loop consists of a single statement that is a loop.
- 121
- 122     • **Pixel**  
123       A pixel, or *picture element*, represents a single element in a digital image. Typically pixels are composed of multiple  
124       color components such as a red, green and blue values. Other color representation exist, including single channel  
125       images that just represent intensity or black and white values.
- 126
- 127     • **Reference counting**  
128       Reference counting is a resource management technique to manage an object’s lifetime. References to an object  
129       are counted and the object is kept alive as long as there is at least one reference to it. A reference counted object  
130       is destroyed when the last reference disappears.
- 131
- 132     • **SIMD unit**  
133       Single Instruction Multiple Data. A machine programming model where a single instruction operates over multiple  
134       pieces of data. Translating a program to use SIMD is known as vectorization. GPUs have multiple SIMD units,  
135       which are the streaming multiprocessors.
- 136     

**Informative:** An SSE (Nehalem, Phenom) or AVX (Sandy Bridge) or LRBni (Larrabee) vector unit is a SIMD unit or  
137       vector processor.
- 138
- 139     • **SMP**  
140       Symmetric Multi-Processor – standard PC multiprocessor architecture.
- 141
- 142     • **Texel**  
143       A texel or *texture element* represents a single element of a texture space. Texel elements are mapped to 1D, 2D or  
144       3D surfaces during sampling, rendering and/or rasterization and end up as pixel elements on a display.
- 145

- 146     • **Texture**  
147       A texture is a 1, 2 or 3 dimensional logical array of texels which is optimized in hardware for spacial access using  
148       texture caches. Textures typically are used to represent image, volumetric or other visual information, although  
149       they are efficient for many data arrays which need to be optimized for spacial access or need to interpolate  
150       between adjacent elements. Textures provide virtualization of storage, whereby shader code can sample a texture  
151       object as if it contained logical elements of one type (e.g., float4) whereas the concrete physical storage of the  
152       texture is represented in terms of a second type (e.g., four 8-bit channels). This allows the application of the same  
153       shader algorithms on different types of concrete data.
- 154
- 155     • **Texture Format**  
156       Texture formats define the type and arrangement of the underlying bytes representing a texel value.  
157       *Informative: Direct3D supports many types of formats, which are described under the DXGI\_FORMAT enumeration.*
- 158
- 159     • **Texture memory**  
160       Texture memory space resides in GPU memory and is cached in texture cache. A texture fetch costs one memory  
161       read from GPU memory only on a cache miss, otherwise it just costs one read from texture cache. The texture  
162       cache is optimized for 2D spatial locality, so threads of the same scheduling unit that read texture addresses that  
163       are close together in 2D will achieve best performance. Also, it is designed for streaming fetches with a constant  
164       latency; a cache hit reduces global memory bandwidth demand but not fetch latency.
- 165
- 166     • **Thread tile**  
167       A set of threads that are scheduled together, can share tile\_static memory, and can participate in barrier  
168       synchronization.
- 169
- 170     • **Tile static memory**  
171       User-managed programmable cache on streaming multiprocessors on GPUs. Shared memory is local to a  
172       multiprocessor and shared across threads executing on the same multiprocessor. Shared memory allocations per  
173       thread group will affect the total number of thread groups that are in-flight per multiprocessor
- 174
- 175     • **Tiling**  
176       Tiling is the partitioning of an N-dimensional dense index space (compute domain) into same sized ‘tiles’ which are  
177       N-dimensional rectangles with sides parallel to the coordinate axes. Tiling is essentially the process of recognizing  
178       the current thread group as being a cooperative gang of threads, with the decomposition of a global index into a  
179       local index plus a tile offset. In C++ AMP it is viewing a global index as a local index and a tile ID described by the  
180       canonical correspondence:  
181            $\text{compute grid} \sim \text{dispatch grid} \times \text{thread group}$   
182       In particular, tiling provides the local geometry with which to take advantage of shared memory and barriers  
183       whose usage patterns enable reducing global memory accesses and coalescing of global memory access. The  
184       former is the most common use of tile\_static memory.
- 185
- 186     • **Restricted function**  
187       A function that is declared to obey the restrictions of a particular C++ AMP subset. A function can be CPU-  
188       restricted, in which case it can run on a host CPU. A function can be amp-restricted, in which case it can run on an  
189       amp-capable accelerator, such as a GPU or VectorCPU. A function can carry more than one restriction.

### 190    1.3 Error Model

191  
192       Host-side runtime library code for C++ AMP has a different error model than device-side code. For more details, examples  
193       and exception categorization see Error Handling.

194  
195       **Host-Side Error Model:** On a host, C++ exceptions and assertions will be used to present semantic errors and hence will be  
196       categorized and listed as error states in API descriptions.

197

198 **Device-Side Error Model:**199 **Microsoft-specific:** The debug\_printf intrinsic is additionally supported for logging messages from within the accelerator  
200 code to the debugger output window.201 **Compile-time asserts:** The C++ intrinsic `static_assert` is often used to handle error states that are detectable at compile  
202 time. In this way `static_assert` is a technique for conveying static semantic errors and as such they will be categorized  
203 similar to exception types.204 

## 1.4 Programming Model

205  
206 The C++ AMP programming model is factored into the following header files:  
207

- 208
- `<amp.h>`
  - `<amp_math.h>`
  - `<amp_graphics.h>`
  - `<amp_short_vectors.h>`

212 C++ AMP programming model is contained in namespace *concurrency* and nested namespaces.  
213

214 Here are the types and patterns that comprise C++ AMP.

- 215
- **Indexing level (`<amp.h>`)**
    - `index<N>`
    - `extent<N>`
    - `tiled_extent<D0,D1,D2>`
    - `tiled_index<D0,D1,D2>`
  - **Data level (`<amp.h>`)**
    - `array<T,N>`
    - `array_view<T,N>, array_view<const T,N>`
    - `copy`
    - `copy_async`
  - **Runtime level (`<amp.h>`)**
    - `accelerator`
    - `accelerator_view`
    - `completion_future`
  - **Call-site level (`<amp.h>`)**
    - `parallel_for_each`
    - copy – various commands to move data between compute nodes
  - **Kernel level (`<amp.h>`)**
    - `tile_barrier`
    - `restrict() clause`
    - `tile_static`
    - Atomic functions
  - **Math functions (`<amp_math.h>`)**
    - Precise math functions
    - Fast math functions
  - **Textures (optional, `<amp_graphics.h>`)**
    - `texture<T,N>`
    - `writeonly_texture_view<T,N> (deprecated)`
    - `texture_view<T,N>`

- ```

244          ○ texture_view<const T, N>
245      • Short vector types (optional, <amp_short_vectors.h>)
246          ○ Short vector types
247      • direct3d interop (optional and Microsoft-specific)
248          ○ Data interoperation on arrays and textures
249          ○ Scheduling interoperation accelerators and accelerator views
250          ○ Direct3D intrinsic functions for clamping, bit counting, and other special arithmetic operations

```

## 2 C++ Language Extensions for Accelerated Computing

C++ AMP adds a closed set<sup>1</sup> of restriction specifiers to the C++ type system, with new syntax, as well as rules for how they behave with respect to conversion rules and overloading.

Restriction specifiers apply to function declarators only. The restriction specifiers perform the following functions:

1. They become part of the signature of the function.
2. They enforce restrictions on the content and/or behaviour of that function.

They may designate a particular subset of the C++ language.

For example, an “amp” restriction would imply that a function must conform to the defined subset of C++ such that it is amenable for use on a typical GPU device.

### 2.1 Syntax

A new grammar production is added to represent a sequence of such restriction specifiers.

```

266      restriction-specifier-seq:
267          restriction-specifier
268          restriction-specifier-seq restriction-specifier
269
270      restriction-specifier:
271          restrict ( restriction-seq )
272
273      restriction-seq:
274          restriction
275          restriction-seq , restriction
276
277      restriction:
278          amp-restriction
279          cpu
280
281      amp-restriction:
282          amp
283

```

The `restrict` keyword is a contextual keyword. The restriction specifiers contained within a `restrict` clause are not reserved words.

Multiple `restrict` clauses, such as `restrict(A) restrict(B)`, behave exactly the same as `restrict(A,B)`. Duplicate restrictions are allowed and behave as if the duplicates are discarded.

---

<sup>1</sup> There is no mechanism proposed here to allow developers to extend the set of restrictions.

290 The `cpu` restriction specifies that this function will be able to run on the host CPU.  
 291  
 292 If a declarator elides the restriction specifier, it behaves as if it were specified with `restrict(cpu)`, except when a restriction  
 293 specifier is determined by the surrounding context as specified in section 2.2.1. If a declarator contains a restriction  
 294 specifier, then it specifies the entire set of restrictions (in other words: `restrict(amp)` means will be able to run on the amp  
 295 target, need not be able to run the CPU).  
 296

### 297 2.1.1 Function Declarator Syntax

298 The function declarator grammar (classic & trailing return type variation) are adjusted as follows:  
 299  
 300     D1 ( parameter-declaration-clause ) cv-qualifier-seq<sub>opt</sub> ref-qualifier<sub>opt</sub> restriction-specifier-seq<sub>opt</sub>  
 301              exception-specification<sub>opt</sub> attribute-specifier<sub>opt</sub>  
 302  
 303     D1 ( parameter-declaration-clause ) cv-qualifier-seq<sub>opt</sub> ref-qualifier<sub>opt</sub> restriction-specifier-seq<sub>opt</sub>  
 304              exception-specification<sub>opt</sub> attribute-specifier<sub>opt</sub> trailing-return-type  
 305

306 Restriction specifiers shall not be applied to other declarators (e.g.: arrays, pointers, references). They can be applied to all  
 307 kinds of functions including free functions, static and non-static member functions, special member functions, and  
 308 overloaded operators.

309 Examples:

```
311
312     auto grod() restrict(amp);
313     auto freedle() restrict(amp)-> double;
314
315     class Fred {
316     public:
317         Fred() restrict(amp)
318             : member-initializer
319             { }
320
321         Fred& operator=(const Fred&) restrict(amp);
322
323         int kreeble(int x, int y) const restrict(amp);
324         static void zot() restrict(amp);
325     };
326
```

327 `restriction-specifier-seqopt` applies to all expressions between the `restriction-specifier-seq` and the end of the function-  
 328 definition, lambda-expression, member-declarator, lambda-declarator or declarator.

### 329 2.1.2 Lambda Expression Syntax

330 The lambda expression syntax is adjusted as follows:

```
331
332     lambda-declarator:
333         ( parameter-declaration-clause ) attribute-specifieropt mutableopt restriction-specifier-seqopt
334         exception-specificationopt trailing-return-typeopt
335
```

336 When a restriction modifier is applied to a lambda expression, the behavior is as if the function call operator of the  
 337 generated closure type is restriction-modified. Implicitly generated special member functions of such closure type follow  
 338 the rules specified in 2.3.2.

339 For example:

```
341
342     Foo ambientVar;
343
344     auto functor = [ambientVar] (int y) restrict(amp) -> int { return y + ambientVar.z; };
345
```

346 is equivalent to:

```

347     Foo ambientVar;
348
349     class <lambdaName> {
350         public:
351             <lambdaName>(const Foo& foo) restrict(amp,cpu)
352                 : capturedFoo(foo)
353             { }
354
355             ~<lambdaName>() restrict(amp,cpu) {}
356
357             int operator()(int y) const restrict(amp) { return y + capturedFoo.z; }
358
359         Foo capturedFoo;
360     };
361
362     <lambdaName> functor;
363

```

### 364 2.1.3 Type Specifiers

365 Restriction specifiers are not allowed anywhere in the type specifier grammar, even if it specifies a function type. For  
 366 example, the following is not well-formed and will produce a syntax error:

```

367     typedef float FuncType(int);
368
369     restrict(cpu) FuncType* pf; // Illegal; restriction specifiers not allowed in type specifiers
370
371

```

372 The correct way to specify the previous example is:

```

373     typedef float FuncType(int) restrict(cpu);
374
375     FuncType* pf;
376
377

```

378 or simply

```

379     float (*pf)(int) restrict(cpu);
380
381

```

## 382 2.2 Meaning of Restriction Specifiers

383 The restriction specifiers on the declaration of a given function *F* must agree with those specified on the definition of  
 384 function *F*.

385 Multiple restriction specifiers may be specified for a given function: the effect is that the function enforces the union of the  
 386 restrictions defined by each restriction modifier.

387 ***Informative:*** *not for this release: It is possible to imagine two restriction specifiers that are intrinsically incompatible with each other (for example, **pure** and **elemental**). When this occurs, the compiler will produce an error.*

388 Refer to section 13 for treatment of versioning of restrictions

393 The restriction specifiers on a function become part of its signature, and thus can be used to overload.

394 Every expression (or sub-expression) that is evaluated in code that has multiple restriction specifiers must have the same  
 395 type in the context of each restriction. It is a compile-time error if an expression can evaluate to different types under the  
 396 different restriction specifiers. Function overloads should be defined with care to avoid a situation where an expression can  
 397 evaluate to different types with different restrictions.

399   **2.2.1 Function Definitions**

400   The restriction specifiers applied to a function definition are recursively applied to all function declarators and type names  
 401   defined within its body that do not have explicit restriction specifiers (i.e.: through nested classes that have member  
 402   functions, and through lambdas.) For example:

```
403
404     void glorp() restrict(amp) {
405         class Foo {
406             void zot() {...} // "zot" is amp-restricted
407         };
408
409         auto f1 = [] (int y) { ... }; // Lambda is amp-restricted
410
411         auto f2 = [] (int y) restrict(cpu) { ... }; // Lambda is cpu-restricted
412
413         typedef int int_void_amp(); // int_void_amp is amp-restricted
414         ...
415     }
```

416   This also applies to the function scope of a lambda body.

418   **2.2.2 Constructors and Destructors**

419   Constructors can have overloads that are differentiated by restriction specifiers.

420

421   Since destructors cannot be overloaded, the destructor must contain a restriction specifier that covers the union of  
 422   restrictions on all the constructors. (A destructor can achieve the same effect of overloading by calling auxiliary cleanup  
 423   functions that have different restriction specifiers.)

424

425   For example:

```
426
427     class Foo {
428     public:
429         Foo() { ... }
430         Foo() restrict(amp) { ... }
431
432         ~Foo() restrict(cpu,amp);
433     };
434
435     void UnrestrictedFunction() {
436         Foo a; // calls "Foo::Foo()"
437
438         ...
439         // a is destructed with "Foo::~Foo()"
440     }
441
442     void RestrictedFunction() restrict(amp) {
443         Foo b; // calls "Foo::Foo() restrict(amp)"
444
445         ...
446         // b is destructed with "Foo::~Foo()"
447     }
448
449     class Bar {
450     public:
451         Bar() { ... }
452         Bar() restrict(amp) { ... }
453
454         ~Bar(); // error: restrict(cpu,amp) required
455     };
456
```

455   A virtual function declaration in a derived class will override a virtual function declaration in a base class only if the derived  
 456   class function has the same restriction specifiers as the base. E.g.:

```
457
458     class Base {
459     public:
```

```

460     virtual void foo() restrict(R1);
461 };
462
463 class Derived : public Base {
464 public:
465     virtual void foo() restrict(R2); // Does not override Base::foo
466 };
467

```

(Note that C++ AMP does not support virtual functions in the current *restrict(amp)* subset.)

## 472 2.3 Expressions Involving Restricted Functions

### 473 2.3.1 Function pointer conversions

474 New implicit conversion rules must be added to account for restricted function pointers (and references). Given an  
 475 expression of type “pointer to R<sub>1</sub>-function”, this type can be implicitly converted to type “pointer to R<sub>2</sub>-function” if and only  
 476 if R<sub>1</sub> has all the restriction specifiers of R<sub>2</sub>. Stated more intuitively, it is okay for the target function to be more restricted  
 477 than the function pointer that invokes it; it’s not okay for it to be less restricted. E.g.:

```

478     int func(int) restrict(R1,R2);
479     int (*pfn)(int) restrict(R1) = func; // ok, since func(int) restrict(R1,R2) is at least R1
480

```

(Note that C++ AMP does not support function pointers in the current *restrict(amp)* subset.)

### 483 2.3.2 Function Overloading

484 Restriction specifiers become part of the function type to which they are attached. I.e.: they become part of the signature  
 485 of the function. Functions can thus be overloaded by differing modifiers, and each unique set of modifiers forms a unique  
 486 overload.

487 The restriction specifiers of a function shall not overlap with any restriction specifiers in another function within the same  
 488 overload set.

```

490
491     int func(int x) restrict(cpu,amp);
492     int func(int x) restrict(cpu); // error, overlaps with previous declaration
493

```

494 The target of the function call operator must resolve to an overloaded set of functions that is *at least* as restricted as the  
 495 body of the calling function (see Overload Resolution). E.g.:

```

496
497     void grod();
498     void glorp() restrict(amp);
499
500     void foo() restrict(amp) {
501         glorp(); // okay: glorp has amp restriction
502         grod(); // error: grod lacks amp restriction
503     }
504

```

505 It is permissible for a less-restrictive call-site to call a more-restrictive function.

506 Compiler-generated constructors and destructors (and other special member functions) behave as if they were declared  
 507 with as many restrictions as possible while avoiding ambiguities and errors. For example:

```

509
510     struct Grod {
511         int a;
512         int b;
513
514         // compiler-generated default constructor: Grod() restrict(cpu,amp);
515

```

```

515
516     int frool() restrict(amp) {
517         return a+b;
518     }
519
520     int blarg() restrict(cpu) {
521         return a*b;
522     }
523
524     // compiler-generated destructor: ~Grod() restrict(cpu,amp);
525 };
526
527 void d3dCaller() restrict(amp) {
528     Grod g; // okay because compiler-generated default constructor is restrict(amp)
529
530     int x = g.frool();
531
532     // g.~Grod() called here; also okay
533 }
534
535 void d3dCaller() restrict(cpu) {
536     Grod g; // okay because compiler-generated default constructor is restrict(cpu)
537
538     int x = g.blarg();
539
540     // g.~Grod() called here; also okay
541 }
542

```

543 The compiler must behave this way since the local usage of “Grod” in this case should not affect other potential uses of it in  
 544 other restricted or unrestricted scopes.

545 More specifically, the compiler follows the standard C++ rules, ignoring restrictions, to determine which special member  
 546 functions to generate and how to generate them. Then the restrictions are set according to the following steps:

547 The compiler sets the restrictions of compiler-generated destructors to the intersection of the restrictions on all of the  
 548 destructors of the data members [*able to destroy all data members*] and all of the base classes’ destructors [*able to call all*  
 549 *base classes’ destructors*]. If there are no such destructors, then all possible restrictions are used [*able to destroy in any*  
 550 *context*]. However, any restriction that would result in an error is not set.

551 The compiler sets the restrictions of compiler-generated default constructors to the intersection of the restrictions on all of  
 552 the default constructors of the member fields [*able to construct all member fields*], all of the base classes’ default  
 553 constructors [*able to call all base classes’ default constructors*], and the destructor of the class [*able to destroy in any*  
 554 *context constructed*]. However, any restriction that would result in an error is not set.

555 The compiler sets the restrictions of compiler-generated copy constructors to the intersection of the restrictions on all of  
 556 the copy constructors of the member fields [*able to construct all member fields*], all of the base classes’ copy constructors  
 557 [*able to call all base classes’ copy constructors*], and the destructor of the class [*able to destroy in any context constructed*].  
 558 However, any restriction that would result in an error is not set.

559 The compiler sets the restrictions of compiler-generated assignment operators to the intersection of the restrictions on all  
 560 of the assignment operators of the member fields [*able to assign all member fields*] and all of the base classes’ assignment  
 561 operators [*able to call all base classes’ assignment operators*]. However, any restriction that would result in an error is not  
 562 set.

### 563 2.3.2.1 Overload Resolution

564 Overload resolution depends on the set of restrictions (function modifiers) in force at the call site.

565

```

572     int func(int x) restrict(A);
573     int func(int x) restrict(B,C);
574     int func(int x) restrict(D);
575
576     void foo() restrict(B) {
577         int x = func(5); // calls func(int x) restrict(B,C)
578         ...
579     }
580

```

581 A call to function *F* is valid if and only if the overload set of *F* covers all the restrictions in force in the calling function. This  
 582 rule can be satisfied by a single function *F* that contains all the required restrictions, or by a set of overloaded functions *F*  
 583 that each specify a subset of the restrictions in force at the call site. For example:

```

584
585     void z() restrict(amp,sse2,cpu) { }
586
587     void z_caller() restrict(amp,sse,cpu) {
588         z(); // okay; all restrictions available in a single function
589     }
590
591     void x() restrict(amp) { }
592     void X() restrict(sse) { }
593     void X() restrict(cpu) { }
594
595     void x_caller() restrict(amp,sse,cpu) {
596         x(); // okay; all restrictions available in separate functions
597     }
598
599     void y() restrict(amp) { }
600
601     void y_caller() restrict(cpu,amp) {
602         y(); // error; no available Y() that satisfies CPU restriction
603     }
604

```

605 When a call to a restricted function is satisfied by more than one function, then the compiler must generate an as-if-  
 606 runtime<sup>3</sup>-dispatch to the correctly restricted version.

### 607 2.3.2.2 Name Hiding

608 Overloading via restriction specifiers does not affect the name hiding rules. For example:

```

609
610     void foo(int x) restrict(amp) { ... }
611
612     namespace N1 {
613         void foo(double d) restrict(cpu) { .... }
614
615         void foo_caller() restrict(amp) {
616             foo(10); // error; global foo() is hidden by N1::foo
617         }
618     }
619

```

620 The name hiding rules in C++11 Section 3.3.10 state that within namespace N1, the global name “Foo” is hidden by the local  
 621 name “Foo”, and is *not overloaded* by it.

### 622 2.3.3 Casting

623 A restricted function type can be cast to a more restricted function type using a normal C-style `cast` or `reinterpret_cast`. (A  
 624 cast is not needed when losing restrictions, only when gaining.) For example:

```

625
626     void unrestricted_func(int,int);
627

```

<sup>2</sup> Note that “sse” is used here for illustration only, and does not imply further meaning to it in this specification.

<sup>3</sup> Compilers are always free to optimize this if they can determine the target statically.

```

628 void restricted_caller() restrict(R) {
629     ((void (*)(int,int))restrict(R)>unrestricted_func)(6, 7);
630     reinterpret_cast<(void (*)(int,int))restrict(R)>(unrestricted_func)(6, 7);
631 }
632

```

633 A program which attempts to invoke a function expression after such unsafe casting can exhibit undefined behavior.

## 634 2.4 amp Restriction Modifier

635 The *amp* restriction modifier applies a relatively small set of restrictions that reflect the current limitations of GPU  
636 hardware and the underlying programming model.

### 637 2.4.1 Restrictions on Types

638 Not all types can be supported on current GPU hardware. The *amp* restriction modifier restricts functions from using  
639 unsupported types, in their function signature or in their function bodies.

640

641 We refer to the set of supported types as being *amp-compatible*. Any type referenced within an amp restriction function  
642 shall be amp-compatible. Some uses require further restrictions.

#### 643 2.4.1.1 Type Qualifiers

644 The *volatile* type qualifier is not supported within an amp-restricted function. A variable or member qualified with *volatile*  
645 may not be declared or accessed in *amp* restricted code.

#### 646 2.4.1.2 Fundamental Types

647 Of the set of C++ fundamental types only the following are supported within an amp-restricted function as *amp-compatible*  
648 types.

649

- 650 • *bool*
- 651 • *int, unsigned int*
- 652 • *long, unsigned long*
- 653 • *float, double*
- 654 • *void*

655 The representation of these types on a device running an *amp* function is identical to that of its host.

656 **Informative:** Floating point types behave the same in amp restricted code as they do in CPU code. C++ AMP imposes the  
657 additional behavioural restriction that an intermediate representation of a floating point expression may not use higher  
658 precision than the operands demand. For example,

```

661 float foo() restrict(amp) {
662     float f1, f2;
663     ...
664     return f1 + f2; // "+" must be performed using "float" precision
665 }
666

```

667 In the above example, the expression “*f1 + f2*” shall not be performed using double (or higher) precision and then converted  
668 back to float.

669

670 **Microsoft-specific:** This is equivalent to the Visual C++ “/fp:precise” mode. C++ AMP does not use higher-precision for  
671 intermediate representations of floating point expressions even when “/fp:fast” is specified.

672    **2.4.1.3 Compound Types**

673    Pointers shall only point to *amp-compatible* types or `concurrency::array` or `concurrency::graphics::texture`. Pointers to  
 674    pointers are not supported. `std::nullptr_t` type is supported and treated as a pointer type. No pointer type is considered  
 675    *amp-compatible*. Pointers are only supported as local variables and/or function parameters and/or function return types.  
 676

677    References (lvalue and rvalue) shall refer only to *amp-compatible* types and/or `concurrency::array` and/or  
 678    `concurrency::graphics::texture`. Additionally, references to pointers are supported as long as the pointer type is itself  
 679    supported. Reference to `std::nullptr_t` is not allowed. No reference type is considered *amp-compatible*. References are only  
 680    supported as local variables and/or function parameters and/or function return types.  
 681

682    `concurrency::array_view` and `concurrency::graphics::writeonly_texture_view` are *amp-compatible* types.  
 683

684    A class type (class, struct, union) is *amp-compatible* if

- 685    • it contains only data members whose types are *amp-compatible*, except for references to instances of classes  
     `array` and `texture`, and
- 686    • the offset of its data members and base classes are at least four bytes aligned, and
- 687    • its data members shall not be bitfields, and
- 688    • it shall not have `virtual` base classes, and `virtual` member functions, and
- 689    • all of its base classes are *amp-compatible*.

691    The element type of an array shall be *amp-compatible* and four byte aligned.  
 692

693    Pointers to members (C++11 8.3.3) shall only refer to non-static data members.  
 694

695    Enumeration types shall have underlying types consisting of `int`, `unsigned int`, `long`, or `unsigned long`.  
 696

697    The representation of an *amp-compatible* compound type (with the exception of pointer & reference) on a device is  
 698    identical to that of its host.

699    **2.4.2 Restrictions on Function Declarators**

700    The function declarator (C++11 8.3.5) of an amp-restricted function:

- 701    • shall not have a trailing ellipsis (...) in its parameter list
- 702    • shall have no parameters, or shall have parameters whose types are *amp-compatible*
- 703    • shall have a return type that is `void` or is *amp-compatible*
- 704    • shall not be `virtual`
- 705    • shall not have a dynamic exception specification
- 706    • shall not have `extern "C"` linkage when multiple restriction specifiers are present

707    **2.4.3 Restrictions on Function Scopes**

708    The function scope of an amp-restricted function may contain any valid C++ declaration, statement, or expression except  
 709    for those which are specified here.

710    **2.4.3.1 Literals**

711    A C++ AMP program is ill-formed if the value of an integer constant or floating point constant exceeds the allowable range  
 712    of any of the above types.

713    **2.4.3.2 Primary Expressions (C++11 5.1)**

714    An identifier or qualified identifier that refers to an object shall refer only to:

- 715    • a parameter to the function, or
- 716    • a local variable declared at a block scope within the function, or
- 717    • a non-static member of the class of which this function is a member, or

- 718     • a *static const* type that can be reduced to a integer literal and is only used as an rvalue, or  
 719     • a global *const* type that can be reduced to a integer literal and is only used as an rvalue, or  
 720     • a captured variable in a lambda expression.

722     

### 2.4.3.3 Lambda Expressions

723     If a lambda expression appears within the body of an amp-restricted function, the *amp* modifier may be elided and the  
 724     lambda is still considered an amp lambda.

725     A lambda expression shall not capture any context variable by reference, except for context variables of type  
 726     *concurrency::array* and *concurrency::graphics::texture*.

729     The effective closure type must be *amp-compatible*.

730     

### 2.4.3.4 Function Calls (C++11 5.2.2)

731     The target of a function call operator:

- 732         • shall not be a virtual function
- 733         • shall not be a pointer to a function
- 734         • shall not recursively invoke itself or any other function that is directly or indirectly recursive.

736     These restrictions apply to all function-like invocations including:

- 737         • object constructors & destructors
- 738         • overloaded operators, including **new** and **delete**.

739     

### 2.4.3.5 Local Declarations

740     Local declarations shall not specify any storage class other than *register*, or *tile\_static*. Variables that are not *tile\_static*  
 741     shall have types that are *amp-compatible*, pointers to *amp-compatible* types, or references to *amp-compatible* types.

742     

#### 2.4.3.5.1 *tile\_static* Variables

743     A variable declared with the *tile\_static* storage class can be accessed by all threads within a tile (group of threads). (The  
 744     *tile\_static* storage class is valid only within a *restrict(amp)* context.) The storage lifetime of a *tile\_static* variable begins  
 745     when the execution of a thread in a tile reaches the point of declaration, and ends when the kernel function is exited by the  
 746     last thread in the tile. Each thread tile accessing the variable shall perceive to access a separate, per-tile, instance of the  
 747     variable.

748     A *tile\_static* variable declaration does not constitute a barrier (see 8.1.1). *tile\_static* variables are not initialized by the  
 749     compiler and assume no default initial values.

751     The *tile\_static* storage class shall only be used to declare local (function or block scope) variables.

753     The type of a *tile\_static* variable or array must be *amp-compatible* and shall not directly or recursively contain any  
 754     concurrency containers (e.g. *concurrency::array\_view*) or reference to concurrency containers.

756     A *tile\_static* variable shall not have an initializer and no constructors or destructors will be called for it; its initial contents  
 757     are undefined.

760     ***Microsoft-specific:* The Microsoft implementation of C++ AMP restricts the total size of *tile\_static* memory to 32K.**

761     

### 2.4.3.6 Type-Casting Restrictions

762     A type-cast shall not be used to convert a pointer to an integral type, nor an integral type to a pointer. This restriction  
 763     applies to *reinterpret\_cast* (C++11 5.2.10) as well as to C-style casts (C++11 5.4).

764  
 765 Casting away *const*-ness may result in a compiler warning and/or undefined behavior.

766 **2.4.3.7 Miscellaneous Restrictions**

767 The pointer-to-member operators `.*` and `->*` shall only be used to access pointer-to-data member objects.

768  
 769 Pointer arithmetic shall not be performed on pointers to *bool* values.

770  
 771 A pointer or reference to an amp-restricted function is not allowed. This is true even outside of an amp-restricted context.

772  
 773 Furthermore, an amp-restricted function shall not contain any of the following:

- 774 • *dynamic\_cast* or *typeid* operators
- 775 • *goto* statements or labeled statements
- 776 • *asm* declarations
- 777 • Function *try* block, *try* blocks, *catch* blocks, or *throw*.

## 778 **3 Device Modeling**

### 780 **3.1 The concept of a compute accelerator**

781 A compute accelerator is a hardware capability that is optimized for data-parallel computing. An accelerator may be a  
 782 device attached to a PCIe bus (such as a GPU), a device integrated on the same die as the GPU, or it might be an extended  
 783 instruction set on the main CPU (such as SSE or AVX).

784  
 785  
 786 *Informative:* Some architectures might bridge these two extremes, such as AMD's Heterogeneous System Architecture  
 787 (AMD HSA) or Intel's Many Integrated Core Architecture (Intel MIC).

788 In the C++ AMP model, an accelerator may have private memory which is not generally accessible by the host. C++ AMP  
 789 allows data to be allocated in the accelerator memory and references to this data may be manipulated on the host, which  
 790 can involve making implicit copies of the data. Likewise, accelerator may reference memory allocted on the host. In some  
 791 cases, accelerator memory and CPU memory are one and the same. And depending upon the architecture, there may  
 792 never be any need to copy between the two physical locations of memory. C++ AMP provides for coding patterns that  
 793 allow the C++ AMP runtime to avoid or perform copies as required.

794  
 795  
 796 C++ AMP has functionality for copying data between host and accelerator memories. A copy from accelerator-to-host is  
 797 always a synchronization point, unless an explicit asynchronous copy is specified. In general, for optimal performance,  
 798 memory content should stay on an accelerator as long as possible.

### 800 **3.2 accelerator**

801 An *accelerator* is an abstraction of a physical data-parallel-optimized compute node. An accelerator is often a GPU, but can  
 802 also be a virtual host-side entity such as the Microsoft DirectX *REF* device, or *WARP* (a CPU-side device accelerated using  
 803 SSE instructions), or can refer to the CPU itself.

#### 804 **3.2.1 Default Accelerator**

805 C++ AMP supports the notion of a default accelerator, an accelerator which is chosen automatically when the program does  
 806 not explicitly do so.

807  
 808 A user may explicitly create a default accelerator object in one of two ways:  
 809

```

810     1. Invoke the default constructor:
811         accelerator def;
812
813     2. Use the default_accelerator device path:
814         accelerator def(accelerator::default_accelerator);
815
816
817

```

818 The user may also influence which accelerator is chosen as the default by calling *accelerator::set\_default* prior to invoking  
 819 any operation which would otherwise choose the default. Such operations include invoking *parallel\_for\_each* without an  
 820 explicit *accelerator\_view* argument, or creating an *array* not bound to an explicit *accelerator\_view*, etc. Note that querying  
 821 or obtaining a default accelerator object does not fix the value for default accelerator; it just allows users to determine  
 822 what the runtime's choice would be before attempting to override it.

823  
 824 If the user does not call *accelerator::set\_default*, the default is chosen in an implementation specific manner.  
 825

826 ***Microsoft-specific:***

827 The Microsoft implementation of C++ AMP uses the the following heuristic to select a default accelerator when one is not  
 828 specified by a call to *accelerator::set\_default*:

- 829 1. If using the debug runtime, prefer an accelerator that supports debugging.
- 830 2. If the process environment variable *CPPAMP\_DEFAULT\_ACCELERATOR* is set, interpret its value as a device path  
     and prefer the device that corresponds to it.
- 831 3. Otherwise, the following criteria are used to determine the 'best' accelerator:
  - 832 a. Prefer non-emulated devices. Among multiple non-emulated devices:
    - 833 i. Prefer the device with the most available memory.
    - 834 ii. Prefer the device which is not attached to the display.
  - 835 b. Among emulated devices, prefer accelerated devices such as WARP over the REF device.

836  
 837     Note that the *cpu\_accelerator* is never considered among the candidates in the above heuristic.

839 **3.2.2 Synopsis**

```

840
841 class accelerator
842 {
843 public:
844     static const wchar_t default_accelerator[]; // = L"default"
845
846     // Microsoft-specific:
847     static const wchar_t direct3d_warp[]; // = L"direct3d\\warp"
848     static const wchar_t direct3d_ref[]; // = L"direct3d\\ref"
849
850     static const wchar_t cpu_accelerator[]; // = L"cpu"
851
852     accelerator();
853     explicit accelerator(const wstring& path);
854     accelerator(const accelerator& other);
855
856     static vector<accelerator> get_all();
857     static bool set_default(const wstring& path);
858     static accelerator_view get_auto_selection_view();
859     accelerator& operator=(const accelerator& other);
860

```

```

861     __declspec(property(get=get_device_path)) wstring device_path;
862     __declspec(property(get=get_version)) unsigned int version; // hiword=major, loword=minor
863     __declspec(property(get=get_description)) wstring description;
864     __declspec(property(get=get_is_debug)) bool is_debug;
865     __declspec(property(get=get_is_emulated)) bool is_emulated;
866     __declspec(property(get=get_has_display)) bool has_display;
867     __declspec(property(get=get_supports_double_precision)) bool supports_double_precision;
868     __declspec(property(get=get_supports_limited_double_precision))
869         bool supports_limited_double_precision;
870     __declspec(property(get=get_dedicated_memory)) size_t dedicated_memory;
871     __declspec(property(get=get_default_view)) accelerator_view default_view;
872     __declspec(property(get=get_default_cpu_access_type)) access_type default_cpu_access_type;
873     __declspec(property(get=get_supports_cpu_shared_memory)) bool supports_cpu_shared_memory;

874     wstring get_device_path() const;
875     unsigned int get_version() const; // hiword=major, loword=minor
876     wstring get_description() const;
877     bool get_is_debug() const;
878     bool get_is_emulated() const;
879     bool get_has_display() const;
880     bool get_supports_double_precision() const;
881     bool get_supports_limited_double_precision() const;
882     size_t dedicated_memory() const;
883     accelerator_view get_default_view() const;
884     access_type get_default_cpu_access_type() const;
885     bool get_supports_cpu_shared_memory() const;
886
887     bool set_default_cpu_access_type(access_type default_cpu_access_type)
888     accelerator_view create_view();
889     accelerator_view create_view(queueing_mode qmode);

890
891     bool operator==(const accelerator& other) const;
892     bool operator!=(const accelerator& other) const;
893 };
894
895

```

### `class accelerator;`

Represents a physical accelerated computing device. An object of this type can be created by enumerating the available devices, or getting the default device.

#### **Microsoft-specific:**

An `accelerator` object can be created by getting the reference device, or the WARP device.

### 896    3.2.3    Static Members

897

#### `static vector<accelerator> accelerator::get_all();`

Returns a `std::vector` of `accelerator` objects (in no specific order) representing all accelerators that are available, including reference accelerators and WARP accelerators if available.

#### **Return Value:**

A vector of accelerators.

898

899

#### `static bool set_default(const wstring& path);`

Sets the default accelerator to the device path identified by the "path" argument. See the constructor "`accelerator(const wstring& path)`" for a description of the allowable path strings.

This establishes a process-wide default accelerator and influences all subsequent operations that might use a default accelerator.

**Parameters**

|             |                                             |
|-------------|---------------------------------------------|
| <i>Path</i> | The device path of the default accelerator. |
|-------------|---------------------------------------------|

**Return Value:**

A Boolean flag indicating whether the default was set. If the default has already been set for this process, this value will be *false*, and the function will have no effect.

900

**static accelerator\_view accelerator::get\_auto\_selection\_view();**

Returns an *accelerator\_view* which when passed as the first argument to a *parallel\_for\_each* call causes the runtime to automatically select the target *accelerator\_view* for executing the *parallel\_for\_each* kernel. In other words, a *parallel\_for\_each* invocation with the *accelerator\_view* returned by *get\_auto\_selection\_view* is the same as a *parallel\_for\_each* invocation without an *accelerator\_view* argument.

For all other purposes, the *accelerator\_view* returned by *get\_auto\_selection\_view* behaves the same as the default *accelerator\_view* of the default accelerator (aka *accelerator().default\_view*).

**Return Value:**

An *accelerator\_view* than can be used to indicate auto selection of the target for a *parallel\_for\_each* execution.

901

## 3.2.4 Constructors

902

**accelerator();**

Constructs a new *accelerator* object that represents the default accelerator. This is equivalent to calling the constructor “*accelerator::default\_accelerator*”.

The actual accelerator chosen as the default can be affected by calling “*accelerator::set\_default*”.

**Parameters:**

*None*.

903

**accelerator(*const wstring& path*);**

Constructs a new *accelerator* object that represents the physical device named by the “path” argument. If the path represents an unknown or unsupported device, an exception will be thrown.

The path can be one of the following:

1. *accelerator::default\_accelerator* (or L“default”), which represents the path of the fastest accelerator available, as chosen by the runtime.
2. *accelerator::cpu\_accelerator* (or L“cpu”), which represents the CPU. Note that *parallel\_for\_each* shall not be invoked over this accelerator.
3. A valid device path that uniquely identifies a hardware accelerator available on the host system.

**Microsoft-specific:**

4. *accelerator::direct3d\_warp* (or L“direct3d\\warp”), which represents the WARP accelerator
5. *accelerator::direct3d\_ref* (or L“direct3d\\ref”), which represents the REF accelerator.

**Parameters:**

|             |                                      |
|-------------|--------------------------------------|
| <i>Path</i> | The device path of this accelerator. |
|-------------|--------------------------------------|

904

**accelerator(*const accelerator& other*);**

Copy constructs an *accelerator* object. This function does a shallow copy with the newly created *accelerator* object pointing to the same underlying device as the passed *accelerator* parameter.

**Parameters:**

|              |                                             |
|--------------|---------------------------------------------|
| <i>Other</i> | The <i>accelerator</i> object to be copied. |
|--------------|---------------------------------------------|

905

907    **3.2.5 Members**

908

```
static const wchar_t default_accelerator[];
static const wchar_t direct3d_warp[];
static const wchar_t direct3d_ref[];
static const wchar_t cpu_accelerator[];
```

These are static constant string literals that represent device paths for known accelerators, or in the case of "default\_accelerator", direct the runtime to choose an accelerator automatically.

**default\_accelerator:** The string L"default" represents the default accelerator, which directs the runtime to choose the fastest accelerator available. The selection criteria are discussed in section 3.2.1 Default Accelerator.

**cpu\_accelerator:** The string L"cpu" represents the host system. This accelerator is used to provide a location for system-allocated memory such as host arrays and staging arrays. It is not a valid target for accelerated computations.

**Microsoft-specific:**

**direct3d\_warp:** The string L"direct3d\\warp" represents the device path of the CPU-accelerated Warp device. On other non-direct3d platforms, this member may not exist.

**direct3d\_ref:** The string L"direct3d\\ref" represents the software rasterizer, or Reference, device. This particular device is useful for debugging. On other non-direct3d platforms, this member may not exist.

909

```
accelerator& operator=(const accelerator& other);
```

Assigns an accelerator object to "this" accelerator object and returns a reference to "this" object. This function does a shallow assignment with the newly created accelerator object pointing to the same underlying device as the passed accelerator parameter.

**Parameters:**

|       |                                             |
|-------|---------------------------------------------|
| Other | The accelerator object to be assigned from. |
|-------|---------------------------------------------|

**Return Value:**

A reference to "this" accelerator object.

910

```
_declspec(property(get=get_default_view)) accelerator_view default_view;
accelerator_view get_default_view() const;
```

Returns the default accelerator view associated with the accelerator. The queuing\_mode of the default accelerator\_view is queuing\_mode\_automatic.

**Return Value:**

The default `accelerator_view` object associated with the accelerator.

911

```
accelerator_view create_view(queuing_mode qmode);
```

Creates and returns a new accelerator view on the accelerator with the supplied queuing mode.

**Return Value:**

The new `accelerator_view` object created on the compute device.

**Parameters:**

|       |                                                                             |
|-------|-----------------------------------------------------------------------------|
| Qmode | The queuing mode of the accelerator_view to be created. See "Queuing Mode". |
|-------|-----------------------------------------------------------------------------|

912

```
accelerator_view create_view();
```

Creates and returns a new accelerator view on the accelerator. Equivalent to "create\_view(queuing\_mode\_automatic)".

**Return Value:**

The new `accelerator_view` object created on the compute device.

913  
914

```
bool operator==(const accelerator& other) const;
```

Compares "this" accelerator with the passed accelerator object to determine if they represent the same underlying device.

**Parameters:**

|              |                                                |
|--------------|------------------------------------------------|
| <i>Other</i> | The accelerator object to be compared against. |
|--------------|------------------------------------------------|

**Return Value:**

A boolean value indicating whether the passed accelerator object is same as "this" accelerator.

915  
916

```
bool operator!=(const accelerator& other) const;
```

Compares "this" accelerator with the passed accelerator object to determine if they represent different devices.

**Parameters:**

|              |                                                |
|--------------|------------------------------------------------|
| <i>Other</i> | The accelerator object to be compared against. |
|--------------|------------------------------------------------|

**Return Value:**

A boolean value indicating whether the passed accelerator object is different from "this" accelerator.

917  
918

```
bool set_default_cpu_access_type(access_type default_cpu_access_type);
```

Sets the default\_cpu\_access\_type for this accelerator.

The default\_cpu\_access\_type is used for arrays created on this accelerator or for implicit array\_view memory allocations accessed on this accelerator.

This method only succeeds if the default\_cpu\_access\_type for the accelerator has not already been overriden by a previous call to this method and the runtime selected default\_cpu\_access\_type for this accelerator has not yet been used for allocating an array or for an implicit array\_view memory allocation on this accelerator.

**Parameters:**

|                                |                                                                                                     |
|--------------------------------|-----------------------------------------------------------------------------------------------------|
| <i>default_cpu_access_type</i> | The default cpu access_type to be used for array/array_view memory allocations on this accelerator. |
|--------------------------------|-----------------------------------------------------------------------------------------------------|

**Return Value:**

A boolean value indicating if the default cpu access\_type for the accelerator was successfully set.

919

### 3.2.6 Properties

The following read-only properties are part of the public interface of the class **accelerator**, to enable querying the accelerator characteristics:

920

```
__declspec(property(get=get_device_path)) wstring device_path;
wstring get_device_path() const;
```

Returns a system-wide unique device instance path that matches the "Device Instance Path" property for the device in Device Manager, or one of the predefined path constants `cpu_accelerator`, `direct3d_warp`, or `direct3d_ref`.

921

```
__declspec(property(get=get_description)) wstring description;
wstring get_description() const;
```

Returns a short textual description of the accelerator device.

922

```
__declspec(property(get=get_version)) unsigned int version;
unsigned int get_version() const;
```

Returns a 32-bit unsigned integer representing the version number of this accelerator. The format of the integer is major.minor, where the major version number is in the high-order 16 bits, and the minor version number is in the low-order bits.

925

```
__declspec(property(get=get_has_display)) bool has_display;
bool get_has_display() const;
```

This property indicates that the accelerator may be shared by (and thus have interference from) the operating system or other system software components for rendering purposes. A C++ AMP implementation may set this property to false should such interference not be applicable for a particular accelerator.

926

```
__declspec(property(get=get_dedicated_memory)) size_t dedicated_memory;
size_t get_dedicated_memory() const;
```

Returns the amount of dedicated memory (in KB) on an accelerator device. There is no guarantee that this amount of memory is actually available to use.

927

```
__declspec(property(get=get_supports_double_precision)) bool supports_double_precision;
bool get_supports_double_precision() const;
```

Returns a Boolean value indicating whether this accelerator supports double-precision (`double`) computations. When this returns true, `supports_limited_double_precision` also returns true.

928

```
__declspec(property(get=get_support_limited_double_precision))
bool supports_limited_double_precision;
bool get_supports_limited_double_precision() const;
```

Returns a boolean value indicating whether the accelerator has limited double precision support (excludes double division, precise\_math functions, int to double, double to int conversions) for a parallel\_for\_each kernel.

929

```
__declspec(property(get=get_is_debug)) bool is_debug;
bool get_is_debug() const;
```

Returns a boolean value indicating whether the accelerator supports debugging.

930

```
__declspec(property(get=get_is_emulated)) bool is_emulated;
bool get_is_emulated() const;
```

Returns a boolean value indicating whether the accelerator is emulated. This is true, for example, with the reference, WARP, and CPU accelerators.

931

```
__declspec(property(get=get_supports_cpu_shared_memory)) bool supports_cpu_shared_memory;
bool get_supports_cpu_shared_memory() const;
```

Returns a boolean value indicating whether the accelerator supports memory accessible both by the accelerator and the CPU.

932

```
__declspec(property(get=get_default_cpu_access_type)) access_type default_cpu_access_type;
access_type get_default_cpu_access_type() const;
```

Get the default cpu access\_type for buffers created on this accelerator

933

### 3.3 accelerator\_view

934

An `accelerator_view` represents a logical view of an accelerator. A single physical compute device may have many logical (isolated) accelerator views. Each accelerator has a default accelerator view and additional accelerator views may be optionally created by the user. Physical devices must potentially be shared amongst many client threads. Client threads may choose to use the same `accelerator_view` of an accelerator or each client may communicate with a compute device via an independent `accelerator_view` object for isolation from other client threads. Work submitted to an `accelerator_view` is guaranteed to be executed in the order that it was submitted; there are no such ordering guarantees for work submitted on different `accelerator_views`.

942

943

An `accelerator_view` can be created with a queuing mode of “immediate” or “automatic”. (See “Queuing Mode”).

944

```

945 3.3.1 Synopsis
946
947 class accelerator_view
948 {
949 public:
950     accelerator_view(const accelerator_view& other);
951
952     accelerator_view& operator=(const accelerator_view& other);
953
954 // Microsoft-specific:
955     __declspec(property(get=get_accelerator)) Concurrency::accelerator accelerator;
956     __declspec(property(get=get_is_debug)) bool is_debug;
957     __declspec(property(get=get_version)) unsigned int version;
958     __declspec(property(get=get_queuing_mode)) queuing_mode queuing_mode;
959     __declspec(property(get=get_is_auto_selection)) bool is_auto_selection;
960
961     accelerator get_accelerator() const;
962     bool get_is_debug() const;
963     unsigned int get_version() const;
964     queuing_mode get_queuing_mode() const;
965     bool get_is_auto_selection() const;
966
967     void flush();
968     void wait();
969     completion_future create_marker();
970
971     bool operator==(const accelerator_view& other) const;
972     bool operator!=(const accelerator_view& other) const;
973 };

```

### class accelerator\_view;

Represents a logical (isolated) accelerator view of a compute accelerator. An object of this type can be obtained by calling the [default\\_view](#) property or [create\\_view](#) member functions on an accelerator object.

974

### 3.3.2 Queuing Mode

An [accelerator\\_view](#) can be created with a queuing mode in one of two states:

```

979     enum queuing_mode {
980         queuing_mode_immediate,
981         queuing_mode_automatic
982     };
983

```

If the queuing mode is [queuing\\_mode\\_immediate](#), then any commands (such as copy or [parallel\\_for\\_each](#)) are sent to the corresponding accelerator before control is returned to the caller.

If the queuing mode is [queuing\\_mode\\_automatic](#), then such commands are queued up on a command queue corresponding to this [accelerator\\_view](#). There are three events that can cause queued commands to be submitted:

- Copying the contents of an array to the host or another accelerator\_view results in all previous commands referencing that array resource (including the copy command itself) to be submitted for execution on the hardware.
- Calling the “accelerator\_view::flush” or “accelerator\_view::wait” methods.

- 993     • The underlying accelerator implementation may internally uses a heuristic to determine when commands are  
 994       submitted to the hardware for execution, for example when resource limits would be exceeded without otherwise  
 995       flushing the queue.

996     

### 3.3.3 Constructors

998 An `accelerator_view` object may only be constructed using a copy or move constructor. There is no default constructor.

|                                                                   |  |
|-------------------------------------------------------------------|--|
| <code>accelerator_view(const accelerator_view&amp; other);</code> |  |
|-------------------------------------------------------------------|--|

999 Copy-constructs an `accelerator_view` object. This function does a shallow copy with the newly created `accelerator_view` object pointing to the same underlying view as the "other" parameter.

|                    |  |
|--------------------|--|
| <b>Parameters:</b> |  |
|--------------------|--|

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <code>other</code> | The <code>accelerator_view</code> object to be copied. |
|--------------------|--------------------------------------------------------|

1001     

### 3.3.4 Members

|                                                                                  |  |
|----------------------------------------------------------------------------------|--|
| <code>accelerator_view&amp; operator=(const accelerator_view&amp; other);</code> |  |
|----------------------------------------------------------------------------------|--|

1002 Assigns an `accelerator_view` object to "this" `accelerator_view` object and returns a reference to "this" object. This function does a shallow assignment with the newly created `accelerator_view` object pointing to the same underlying view as the passed `accelerator_view` parameter.

|                    |  |
|--------------------|--|
| <b>Parameters:</b> |  |
|--------------------|--|

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <code>other</code> | The <code>accelerator_view</code> object to be assigned from. |
|--------------------|---------------------------------------------------------------|

|                      |  |
|----------------------|--|
| <b>Return Value:</b> |  |
|----------------------|--|

|                                                             |  |
|-------------------------------------------------------------|--|
| A reference to "this" <code>accelerator_view</code> object. |  |
|-------------------------------------------------------------|--|

|                                                                                   |  |
|-----------------------------------------------------------------------------------|--|
| <code>_declspec(property(get=get_queuing_mode)) queuing_mode queuing_mode;</code> |  |
|-----------------------------------------------------------------------------------|--|

|                                                     |  |
|-----------------------------------------------------|--|
| <code>queuing_mode get_queuing_mode() const;</code> |  |
|-----------------------------------------------------|--|

1003 Returns the queuing mode that this `accelerator_view` was created with. See "Queuing Mode".

|                      |  |
|----------------------|--|
| <b>Return Value:</b> |  |
|----------------------|--|

|                   |  |
|-------------------|--|
| The queuing mode. |  |
|-------------------|--|

|                                                                                     |  |
|-------------------------------------------------------------------------------------|--|
| <code>_declspec(property(get=get_is_auto_selection)) bool is_auto_selection;</code> |  |
|-------------------------------------------------------------------------------------|--|

|                                                  |  |
|--------------------------------------------------|--|
| <code>bool get_is_auto_selection() const;</code> |  |
|--------------------------------------------------|--|

1004 Returns a boolean value indicating whether the accelerator view when passed to a `parallel_for_each` would result in automatic selection of an appropriate execution target by the runtime. In other words, this is the accelerator view that will be automatically selected if `parallel_for_each` is invoked without explicitly specifying an accelerator view.

|                      |  |
|----------------------|--|
| <b>Return Value:</b> |  |
|----------------------|--|

|                                                                                                                       |  |
|-----------------------------------------------------------------------------------------------------------------------|--|
| A boolean value indicating if the <code>accelerator_view</code> is the auto selection <code>accelerator_view</code> . |  |
|-----------------------------------------------------------------------------------------------------------------------|--|

|                                                                         |  |
|-------------------------------------------------------------------------|--|
| <code>_declspec(property(get=get_version)) unsigned int version;</code> |  |
|-------------------------------------------------------------------------|--|

|                                                |  |
|------------------------------------------------|--|
| <code>unsigned int get_version() const;</code> |  |
|------------------------------------------------|--|

|                                                                                                                                                                                                                                                                         |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Returns a 32-bit unsigned integer representing the version number of this <code>accelerator_view</code> . The format of the integer is major.minor, where the major version number is in the high-order 16 bits, and the minor version number is in the low-order bits. |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|

|                                                                                                                       |  |
|-----------------------------------------------------------------------------------------------------------------------|--|
| The version of the <code>accelerator_view</code> is usually the same as that of the parent <code>accelerator</code> . |  |
|-----------------------------------------------------------------------------------------------------------------------|--|

|                                                                                                                                                                                                       |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <b>Microsoft-specific:</b> The version may differ from the <code>accelerator</code> only when the <code>accelerator_view</code> is created from a <code>direct3d</code> device using the interop API. |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|

1006

```
__declspec(property(get=get_accelerator)) Concurrency::accelerator accelerator;
accelerator_get_accelerator() const;
```

Returns the accelerator that this accelerator\_view has been created on.

1007

```
__declspec(property(get=get_is_debug)) bool is_debug;
bool get_is_debug() const;
```

Returns a boolean value indicating whether the accelerator\_view supports debugging through extensive error reporting.

The is\_debug property of the accelerator view is usually same as that of the parent accelerator.

**Microsoft-specific:** The is\_debug value may differ from the accelerator only when the accelerator\_view is created from a direct3d device using the interop API.

1008

```
void wait();
```

Performs a blocking wait for completion of all commands submitted to the accelerator view prior to calling [wait](#).

**Return Value:**

None

1009

```
void flush();
```

Sends the queued up commands in the accelerator\_view to the device for execution.

An accelerator\_view internally maintains a buffer of commands such as data transfers between the host memory and device buffers, and kernel invocations (parallel\_for\_each calls)). This member function sends the commands to the device for processing. Normally, these commands are sent to the GPU automatically whenever the runtime determines that they need to be, such as when the command buffer is full or when waiting for transfer of data from the device buffers to host memory. The [flush](#) member function will send the commands manually to the device.

Calling this member function incurs an overhead and must be used with discretion. A typical use of this member function would be when the CPU waits for an arbitrary amount of time and would like to force the execution of queued device commands in the meantime. It can also be used to ensure that resources on the accelerator are reclaimed after all references to them have been removed.

Because [flush](#) operates asynchronously, it can return either before or after the device finishes executing the buffered commands. However, the commands will eventually always complete.

If the [queueing\\_mode](#) is [queueing\\_mode\\_immediate](#), this function does nothing.

**Return Value:**

None

1010

```
completion_future create_marker();
```

This command inserts a marker event into the accelerator\_view's command queue. This marker is returned as a [completion\\_future](#) object. When all commands that were submitted prior to the marker event creation have completed, the future is ready.

**Return Value:**

A future which can be waited on, and will block until the current batch of commands has completed.

1011

1012

```
bool operator==(const accelerator_view& other) const;
```

Compares "this" accelerator\_view with the passed accelerator\_view object to determine if they represent the same underlying object.

|                                                                                                           |                                                     |
|-----------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| <b>Parameters:</b>                                                                                        |                                                     |
| Other                                                                                                     | The accelerator_view object to be compared against. |
| <b>Return Value:</b>                                                                                      |                                                     |
| A boolean value indicating whether the passed accelerator_view object is same as "this" accelerator_view. |                                                     |

1013

```
bool operator!=(const accelerator_view& other) const;
```

Compares "this" accelerator\_view with the passed accelerator\_view object to determine if they represent different underlying objects.

1014

|                    |  |
|--------------------|--|
| <b>Parameters:</b> |  |
|--------------------|--|

|       |                                                     |
|-------|-----------------------------------------------------|
| Other | The accelerator_view object to be compared against. |
|-------|-----------------------------------------------------|

|                      |  |
|----------------------|--|
| <b>Return Value:</b> |  |
|----------------------|--|

|                                                                                                                  |
|------------------------------------------------------------------------------------------------------------------|
| A boolean value indicating whether the passed accelerator_view object is different from "this" accelerator_view. |
|------------------------------------------------------------------------------------------------------------------|

1015

### 3.4 Device enumeration and selection API

1016

The physical compute devices can be enumerated or selected by calling the following static member function of the class accelerator.

1017

```
vector<accelerator> accelerator::get_all();
```

1018

As an example, if one wants to find an accelerator that is not emulated and is not attached to a display, one could do the following:

1019

```
vector<accelerator> gpus = accelerator::get_all();
auto headlessIter = std::find_if(gpus.begin(), gpus.end(), [] (accelerator& acc1) {
    return !acc1.has_display && !acc1.is_emulated;
});
```

1020

1021

## 4 Basic Data Elements

1022

C++ AMP enables programmers to express solutions to data-parallel problems in terms of N-dimensional data aggregates and operations over them.

1023

Fundamental to C++ AMP is the concept of an array. An array associates values in an index space with an element type. For example an array could be the set of pixels on a screen where each pixel is represented by four 32-bit values: [Red](#), [Green](#), [Blue](#) and [Alpha](#). The index space would then be the screen resolution, for example all points:

```
{ {y, x} | 0 <= y < 1200, 0 <= x < 1600, x and y are integers }.
```

1024

### 4.1 index<N>

1025

Defines an N-dimensional index point; which may also be viewed as a vector based at the origin in N-space.

1026

The index<N> type represents an N-dimensional vector of [int](#) which specifies a unique position in an N-dimensional space. The dimensions in the coordinate vector are ordered from most-significant to least-significant. Thus, in Cartesian 3-dimensional space, where a common convention exists that the Z dimension (plane) is most significant, the Y dimension (row) is second in significance and the X dimension (column) is the least significant, the index vector (2,0,4) represents the position at (Z=2, Y=0, X=4).

1047  
 1048 The position is relative to the origin in the N-dimensional space, and can contain negative component values.  
 1049  
 1050 ***Informative:*** As a scoping decision, it was decided to limit specializations of index, extent, etc. to 1, 2, and 3 dimensions.  
 1051 This also applies to arrays and array\_views. General N-dimensional support is still provided with slightly reduced  
 1052 convenience.  
 1053

#### 1054 4.1.1 Synopsis

```
1055
1056 template <int N>
1057 class index {
1058 public:
1059     static const int rank = N;
1060     typedef int value_type;
1061
1062     index() restrict(amp,cpu);
1063     index(const index& other) restrict(amp,cpu);
1064     explicit index(int i0) restrict(amp,cpu); // N==1
1065     index(int i0, int i1) restrict(amp,cpu); // N==2
1066     index(int i0, int i1, int i2) restrict(amp,cpu); // N==3
1067     explicit index(const int components[N]) restrict(amp,cpu);
1068
1069     index& operator=(const index& other) restrict(amp,cpu);
1070
1071     int operator[](unsigned int c) const restrict(amp,cpu);
1072     int& operator[](unsigned int c) restrict(amp,cpu);
1073
1074     template <int N>
1075         friend bool operator==(const index<N>& lhs, const index<N>& rhs) restrict(amp,cpu);
1076     template <int N>
1077         friend bool operator!=(const index<N>& lhs, const index<N>& rhs) restrict(amp,cpu);
1078     template <int N>
1079         friend index<N> operator+(const index<N>& lhs,
1080                                     const index<N>& rhs) restrict(amp,cpu);
1081     template <int N>
1082         friend index<N> operator-(const index<N>& lhs,
1083                                     const index<N>& rhs) restrict(amp,cpu);
1084
1085     index& operator+=(const index& rhs) restrict(amp,cpu);
1086     index& operator-=(const index& rhs) restrict(amp,cpu);
1087
1088     template <int N>
1089         friend index<N> operator+(const index<N>& lhs, int rhs) restrict(amp,cpu);
1090     template <int N>
1091         friend index<N> operator+(int lhs, const index<N>& rhs) restrict(amp,cpu);
1092     template <int N>
1093         friend index<N> operator-(const index<N>& lhs, int rhs) restrict(amp,cpu);
1094     template <int N>
1095         friend index<N> operator-(int lhs, const index<N>& rhs) restrict(amp,cpu);
1096     template <int N>
1097         friend index<N> operator*(const index<N>& lhs, int rhs) restrict(amp,cpu);
1098     template <int N>
1099         friend index<N> operator*(int lhs, const index<N>& rhs) restrict(amp,cpu);
1100     template <int N>
1101         friend index<N> operator/(const index<N>& lhs, int rhs) restrict(amp,cpu);
1102     template <int N>
```

```

1103     friend index<N> operator/(int lhs, const index<N>& rhs) restrict(amp,cpu);
1104     template <int N>
1105         friend index<N> operator%(const index<N>& lhs, int rhs) restrict(amp,cpu);
1106     template <int N>
1107         friend index<N> operator%=(int lhs, const index<N>& rhs) restrict(amp,cpu);
1108
1109     index& operator+=(int rhs) restrict(amp,cpu);
1110     index& operator-=(int rhs) restrict(amp,cpu);
1111     index& operator*=(int rhs) restrict(amp,cpu);
1112     index& operator/=(int rhs) restrict(amp,cpu);
1113     index& operator%=(int rhs) restrict(amp,cpu);
1114
1115     index& operator++() restrict(amp,cpu);
1116     index operator++(int) restrict(amp,cpu);
1117     index& operator--() restrict(amp,cpu);
1118     index operator--(int) restrict(amp,cpu);
1119 };
1120
1121
1122 
```

**template <int N> class index;**

Represents a unique position in N-dimensional space.

**Template Arguments**

|          |                                                                                                                                                                                     |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>N</i> | The dimensionality space into which this index applies. Special constructors are supplied for the cases where $N \in \{1, 2, 3\}$ , but <i>N</i> can be any integer greater than 0. |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

1123

**static const int rank = N;**

A static member of `index<N>` that contains the rank of this index.

1124

**typedef int value\_type;**

The element type of `index<N>`.

1125

1126

1127

**4.1.2 Constructors****index() restrict(amp,cpu);**

Default constructor. The value at each dimension is initialized to zero. Thus, `"index<3> ix;"` initializes the variable to the position (0,0,0).

1128

1129

**index(const index& other) restrict(amp,cpu);**

Copy constructor. Constructs a new `index<N>` from the supplied argument "other".

**Parameters:**

|              |                                                                                        |
|--------------|----------------------------------------------------------------------------------------|
| <i>other</i> | An object of type <code>index&lt;N&gt;</code> from which to initialize this new index. |
|--------------|----------------------------------------------------------------------------------------|

1130

**explicit index(int i0) restrict(amp,cpu); // N==1  
index(int i0, int i1) restrict(amp,cpu); // N==2  
index(int i0, int i1, int i2) restrict(amp,cpu); // N==3**

Constructs an `index<N>` with the coordinate values provided by  $i_{0..2}$ . These are specialized constructors that are only valid when the rank of the index  $N \in \{1, 2, 3\}$ . Invoking a specialized constructor whose argument count  $\neq N$  will result in a compilation error.

**Parameters:**

|                                         |                                           |
|-----------------------------------------|-------------------------------------------|
| <i>i0</i> [, <i>i1</i> [, <i>i2</i> ] ] | The component values of the index vector. |
|-----------------------------------------|-------------------------------------------|

1131

```
explicit index(const int components[N]) restrict(amp,cpu);
```

Constructs an `index<N>` with the coordinate values provided by the array of `int` component values. If the coordinate array length  $\neq N$ , the behavior is undefined. If the array value is NULL or not a valid pointer, the behavior is undefined.

**Parameters:**

|                         |                                          |
|-------------------------|------------------------------------------|
| <code>components</code> | An array of $N$ <code>int</code> values. |
|-------------------------|------------------------------------------|

1132

1133

```
index& operator=(const index& other) restrict(amp,cpu);
```

Assigns the component values of "other" to this `index<N>` object.

**Parameters:**

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <code>other</code> | An object of type <code>index&lt;N&gt;</code> from which to copy into this index. |
|--------------------|-----------------------------------------------------------------------------------|

**Return Value:**

Returns `*this`.

1134

```
int operator[](unsigned int c) const restrict(amp,cpu);
int& operator[](unsigned int c) restrict(amp,cpu);
```

Returns the index component value at position `c`.

**Parameters:**

|                |                                                        |
|----------------|--------------------------------------------------------|
| <code>c</code> | The dimension axis whose coordinate is to be accessed. |
|----------------|--------------------------------------------------------|

**Return Value:**

A the component value at position `c`.

1135

**4.1.4 Operators**

1136

```
template <int N>
friend bool operator==(const index<N>& lhs, const index<N>& rhs) restrict(amp,cpu);
template <int N>
friend bool operator!=(const index<N>& lhs, const index<N>& rhs) restrict(amp,cpu);
```

Compares two objects of `index<N>`.

The expression

`leftIdx  $\oplus$  rightIdx`

is true if `leftIdx[i]  $\oplus$  rightIdx[i]` for every  $i$  from 0 to  $N-1$ .

**Parameters:**

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <code>lhs</code> | The left-hand <code>index&lt;N&gt;</code> to be compared. |
|------------------|-----------------------------------------------------------|

|                  |                                                            |
|------------------|------------------------------------------------------------|
| <code>rhs</code> | The right-hand <code>index&lt;N&gt;</code> to be compared. |
|------------------|------------------------------------------------------------|

1137

```
template <int N>
friend index<N> operator+(const index<N>& lhs, const index<N>& rhs) restrict(amp,cpu);
template <int N>
friend index<N> operator-(const index<N>& lhs, const index<N>& rhs) restrict(amp,cpu);
```

Binary arithmetic operations that produce a new `index<N>` that is the result of performing the corresponding pair-wise binary arithmetic operation on the elements of the operands. The `result index<N>` is such that for a given operator  $\oplus$ ,  
 $result[i] = leftIdx[i] \oplus rightIdx[i]$

for every  $i$  from 0 to  $N-1$ .

**Parameters:**

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| <code>lhs</code> | The left-hand <code>index&lt;N&gt;</code> of the arithmetic operation. |
|------------------|------------------------------------------------------------------------|

|                  |                                                                         |
|------------------|-------------------------------------------------------------------------|
| <code>rhs</code> | The right-hand <code>index&lt;N&gt;</code> of the arithmetic operation. |
|------------------|-------------------------------------------------------------------------|

1138

**Parameters:**

1139

```
index& operator+=(const index& rhs) restrict(amp,cpu);
index& operator-=(const index& rhs) restrict(amp,cpu);
```

For a given operator  $\oplus$ , produces the same effect as

`(*this) = (*this)  $\oplus$  rhs;`

The return value is “\*this”.

**Parameters:**

|                  |                                                                         |
|------------------|-------------------------------------------------------------------------|
| <code>rhs</code> | The right-hand <code>index&lt;N&gt;</code> of the arithmetic operation. |
|------------------|-------------------------------------------------------------------------|

1140

1141

```
template <int N>
    friend index<N> operator+(const index<N>& idx, int value) restrict(amp,cpu);
template <int N>
    friend index<N> operator+(int value, const index<N>& idx) restrict(amp,cpu);
template <int N>
    friend index<N> operator-(const index<N>& idx, int value) restrict(amp,cpu);
template <int N>
    friend index<N> operator-(int value, const index<N>& idx) restrict(amp,cpu);
template <int N>
    friend index<N> operator*(const index<N>& idx, int value) restrict(amp,cpu);
template <int N>
    friend index<N> operator*(int value, const index<N>& idx) restrict(amp,cpu);
template <int N>
    friend index<N> operator/(const index<N>& idx, int value) restrict(amp,cpu);
template <int N>
    friend index<N> operator/(int value, const index<N>& idx) restrict(amp,cpu);
template <int N>
    friend index<N> operator%(const index<N>& idx, int value) restrict(amp,cpu);
template <int N>
    friend index<N> operator%(int value, const index<N>& idx) restrict(amp,cpu);
```

Binary arithmetic operations that produce a new `index<N>` that is the result of performing the corresponding binary arithmetic operation on the elements of the index operands. The `result index<N>` is such that for a given operator  $\oplus$ ,

$$\text{result}[i] = \text{idx}[i] \oplus \text{value}$$

or

$$\text{result}[i] = \text{value} \oplus \text{idx}[i]$$

for every  $i$  from 0 to  $N-1$ .

**Parameters:**

|                  |                                         |
|------------------|-----------------------------------------|
| <code>idx</code> | The <code>index&lt;N&gt;</code> operand |
|------------------|-----------------------------------------|

|                    |                     |
|--------------------|---------------------|
| <code>value</code> | The integer operand |
|--------------------|---------------------|

1142

```
index& operator+=(int value) restrict(amp,cpu);
index& operator-=(int value) restrict(amp,cpu);
index& operator*=(int value) restrict(amp,cpu);
index& operator/=(int value) restrict(amp,cpu);
index& operator%=(int value) restrict(amp,cpu);
```

For a given operator  $\oplus$ , produces the same effect as

$$(*\text{this}) = (*\text{this}) \oplus \text{value};$$

The return value is “\*this”.

**Parameters:**

|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <code>value</code> | The right-hand <code>int</code> of the arithmetic operation. |
|--------------------|--------------------------------------------------------------|

1143

1144

```
index& operator++() restrict(amp,cpu);
index operator++(int) restrict(amp,cpu);
index& operator--() restrict(amp,cpu);
index operator--(int) restrict(amp,cpu);
```

For a given operator  $\oplus$ , produces the same effect as

$$(*\text{this}) = (*\text{this}) \oplus 1;$$

For prefix increment and decrement, the return value is “\*this”. Otherwise a new `index<N>` is returned.

1145

1146 **4.2 extent<N>**

1147  
 1148 The extent<N> type represents an N-dimensional vector of *int* which specifies the bounds of an N-dimensional space with  
 1149 an origin of 0. The values in the coordinate vector are ordered from most-significant to least-significant. Thus, in Cartesian  
 1150 3-dimensional space, where a common convention exists that the Z dimension (plane) is most significant, the Y dimension  
 1151 (row) is second in significance and the X dimension (column) is the least significant, the extent vector (7,5,3) represents a  
 1152 space where the Z coordinate ranges from 0 to 6, the Y coordinate ranges from 0 to 4, and the X coordinate ranges from 0  
 1153 to 2.

1154 **4.2.1 Synopsis**

```
1155
1156 template <int N>
1157 class extent {
1158 public:
1159     static const int rank = N;
1160     typedef int value_type;
1161
1162     extent() restrict(amp,cpu);
1163     extent(const extent& other) restrict(amp,cpu);
1164     explicit extent(int e0) restrict(amp,cpu); // N==1
1165     extent(int e0, int e1) restrict(amp,cpu); // N==2
1166     extent(int e0, int e1, int e2) restrict(amp,cpu); // N==3
1167     explicit extent(const int components[N]) restrict(amp,cpu);
1168
1169     extent& operator=(const extent& other) restrict(amp,cpu);
1170
1171     int operator[](unsigned int c) const restrict(amp,cpu);
1172     int& operator[](unsigned int c) restrict(amp,cpu);
1173
1174     unsigned int size() const restrict(amp,cpu);
1175
1176     bool contains(const index<N>& idx) const restrict(amp,cpu);
1177
1178     template <int D0>           tiled_extent<D0> tile() const restrict(amp,cpu);
1179     template <int D0, int D1>    tiled_extent<D0,D1> tile() const restrict(amp,cpu);
1180     template <int D0, int D1, int D2> tiled_extent<D0,D1,D2> tile() const restrict(amp,cpu);
1181
1182     extent operator+(const index<N>& idx) restrict(amp,cpu);
1183     extent operator-(const index<N>& idx) restrict(amp,cpu);
1184
1185     extent& operator+=(const index<N>& idx) restrict(amp,cpu);
1186     extent& operator-=(const index<N>& idx) restrict(amp,cpu);
1187     extent& operator+=(const extent& ext) restrict(amp,cpu);
1188     extent& operator-=(const extent& ext) restrict(amp,cpu);
1189
1190     template <int N>
1191         friend extent<N> operator+(const extent<N>& lhs,
1192                                     const extent<N>& rhs) restrict(amp,cpu);
1193     template <int N>
1194         friend extent<N> operator-(const extent<N>& lhs,
1195                                     const extent<N>& rhs) restrict(amp,cpu);
1196
1197     template <int N>
1198         friend bool operator==(const extent<N>& lhs, const extent<N>& rhs) restrict(amp,cpu);
1199     template <int N>
1200         friend bool operator!=(const extent<N>& lhs, const extent<N>& rhs) restrict(amp,cpu);
1201
```

```

1202     template <int N>
1203         friend extent<N> operator+(const extent<N>& lhs, int rhs) restrict(amp,cpu);
1204     template <int N>
1205         friend extent<N> operator+(int lhs, const extent<N>& rhs) restrict(amp,cpu);
1206     template <int N>
1207         friend extent<N> operator-(const extent<N>& lhs, int rhs) restrict(amp,cpu);
1208     template <int N>
1209         friend extent<N> operator-(int lhs, const extent<N>& rhs) restrict(amp,cpu);
1210     template <int N>
1211         friend extent<N> operator*(const extent<N>& lhs, int rhs) restrict(amp,cpu);
1212     template <int N>
1213         friend extent<N> operator*(int lhs, const extent<N>& rhs) restrict(amp,cpu);
1214     template <int N>
1215         friend extent<N> operator/(const extent<N>& lhs, int rhs) restrict(amp,cpu);
1216     template <int N>
1217         friend extent<N> operator/(int lhs, const extent<N>& rhs) restrict(amp,cpu);
1218     template <int N>
1219         friend extent<N> operator%(const extent<N>& lhs, int rhs) restrict(amp,cpu);
1220     template <int N>
1221         friend extent<N> operator%(int lhs, const extent<N>& rhs) restrict(amp,cpu);
1222
1223 extent& operator+=(int rhs) restrict(amp,cpu);
1224 extent& operator-=(int rhs) restrict(amp,cpu);
1225 extent& operator*=(int rhs) restrict(amp,cpu);
1226 extent& operator/=(int rhs) restrict(amp,cpu);
1227 extent& operator%=(int rhs) restrict(amp,cpu);
1228
1229 extent& operator++() restrict(amp,cpu);
1230 extent operator++(int) restrict(amp,cpu);
1231 extent& operator--() restrict(amp,cpu);
1232 extent operator--(int) restrict(amp,cpu);
1233 };
1234
1235

```

**template <int N> class extent;**

Represents a unique position in N-dimensional space.

**Template Arguments**

|          |                                                                                                                                                                                                                                          |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>N</i> | The dimension to this extent applies. Special constructors are supplied for the cases where $N \in \{ 1,2,3 \}$ , but <i>N</i> can be any integer greater than or equal to 1.<br><b>Microsoft-specific:</b> <i>N</i> can not exceed 128. |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**static const int rank = N;**

A static member of `extent<N>` that contains the rank of this extent.

**typedef int value\_type;**

The element type of `extent<N>`.

1239

#### 4.2.2 Constructors

**extent() restrict(amp,cpu);**

Default constructor. The value at each dimension is initialized to zero. Thus, `"extent<3> ix;"` initializes the variable to the position (0,0,0).

**Parameters:**

None.

1240

1241

```
extent(const extent& other) restrict(amp,cpu);
```

Copy constructor. Constructs a new `extent<N>` from the supplied argument `ix`.

**Parameters:**

|                    |                                                                                          |
|--------------------|------------------------------------------------------------------------------------------|
| <code>other</code> | An object of type <code>extent&lt;N&gt;</code> from which to initialize this new extent. |
|--------------------|------------------------------------------------------------------------------------------|

1242

```
explicit extent(int e0) restrict(amp,cpu); // N==1
extent(int e0, int e1) restrict(amp,cpu); // N==2
extent(int e0, int e1, int e2) restrict(amp,cpu); // N==3
```

Constructs an `extent<N>` with the coordinate values provided by `e0...2`. These are specialized constructors that are only valid when the rank of the extent  $N \in \{1,2,3\}$ . Invoking a specialized constructor whose argument count  $\neq N$  will result in a compilation error.

**Parameters:**

|                                |                                            |
|--------------------------------|--------------------------------------------|
| <code>e0 [, e1 [, e2 ]]</code> | The component values of the extent vector. |
|--------------------------------|--------------------------------------------|

1243

```
explicit extent(const int components[N]) restrict(amp,cpu);
```

Constructs an `extent<N>` with the coordinate values provided the array of `int` component values. If the coordinate array length  $\neq N$ , the behavior is undefined. If the array value is NULL or not a valid pointer, the behavior is undefined.

**Parameters:**

|                         |                                          |
|-------------------------|------------------------------------------|
| <code>components</code> | An array of $N$ <code>int</code> values. |
|-------------------------|------------------------------------------|

1244

## 4.2.3 Members

1245

```
extent& operator=(const extent& other) restrict(amp,cpu);
```

Assigns the component values of “`other`” to this `extent<N>` object.

**Parameters:**

|                    |                                                                                     |
|--------------------|-------------------------------------------------------------------------------------|
| <code>other</code> | An object of type <code>extent&lt;N&gt;</code> from which to copy into this extent. |
|--------------------|-------------------------------------------------------------------------------------|

**Return Value:**

Returns `*this`.

1246

```
int operator[](unsigned int c) const restrict(amp,cpu);
int& operator[](unsigned int c) restrict(amp,cpu);
```

Returns the extent component value at position `c`.

**Parameters:**

|                |                                                        |
|----------------|--------------------------------------------------------|
| <code>c</code> | The dimension axis whose coordinate is to be accessed. |
|----------------|--------------------------------------------------------|

**Return Value:**

A the component value at position `c`.

1247

```
bool contains(const index<N>& idx) const restrict(amp,cpu);
```

Tests whether the index “`idx`” is properly contained within this extent (with an assumed origin of zero).

**Parameters:**

|                  |                                               |
|------------------|-----------------------------------------------|
| <code>idx</code> | An object of type <code>index&lt;N&gt;</code> |
|------------------|-----------------------------------------------|

**Return Value:**

Returns `true` if the “`idx`” is contained within the space defined by this extent (with an assumed origin of zero).

1248

```
unsigned int size() const restrict(amp,cpu);
```

This member function returns the total linear size of this `extent<N>` (in units of elements), which is computed as:

$$\text{extent}[0] * \text{extent}[1] \dots * \text{extent}[N-1]$$

1249

```
template <int D0> tiled_extent<D0> tile() const restrict(amp,cpu);
template <int D0, int D1> tiled_extent<D0,D1> tile() const restrict(amp,cpu);
template <int D0, int D1, int D2> tiled_extent<D0,D1,D2> tile() const restrict(amp,cpu);
```

Produces a `tiled_extent` object with the tile extents given by `D0`, `D1`, and `D2`.

1250

`tile<D0,D1,D2>()` is only supported on `extent<3>`. It will produce a compile-time error if used on an `extent` where  $N \neq 3$ .  
`tile<D0,D1>()` is only supported on `extent <2>`. It will produce a compile-time error if used on an `extent` where  $N \neq 2$ .  
`tile<D0>()` is only supported on `extent <1>`. It will produce a compile-time error if used on an `extent` where  $N \neq 1$ .

1251

1252 **4.2.4 Operators**

1253

```
template <int N>
    friend extent<N> operator+(const extent<N>& lhs, const extent<N>& rhs) restrict(amp,cpu);
template <int N>
    friend extent<N> operator-(const extent<N>& lhs, const extent<N>& rhs) restrict(amp,cpu);
```

Adds (or subtracts) two objects of `extent<N>` to form a new extent. The `result extent<N>` is such that for a given operator  $\oplus$ ,  

$$\text{result}[i] = \text{leftExt}[i] \oplus \text{rightExt}[i]$$

for every  $i$  from 0 to  $N-1$ .

**Parameters:**

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| <code>lhs</code> | The left-hand <code>extent&lt;N&gt;</code> to be compared.  |
| <code>rhs</code> | The right-hand <code>extent&lt;N&gt;</code> to be compared. |

1254

1255

```
template <int N>
    friend bool operator==(const extent<N>& lhs, const extent<N>& rhs) restrict(amp,cpu);
template <int N>
    friend bool operator!=(const extent<N>& lhs, const extent<N>& rhs) restrict(amp,cpu);
```

Compares two objects of `extent<N>`.

The expression

$\text{leftExt} \oplus \text{rightExt}$

is true if  $\text{leftExt}[i] \oplus \text{rightExt}[i]$  for every  $i$  from 0 to  $N-1$ .

**Parameters:**

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| <code>lhs</code> | The left-hand <code>extent&lt;N&gt;</code> to be compared.  |
| <code>rhs</code> | The right-hand <code>extent&lt;N&gt;</code> to be compared. |

1256

```
extent<N>& operator+=(const extent<N>& ext) restrict(amp,cpu);
extent<N>& operator-=(const extent<N>& ext) restrict(amp,cpu);
```

Adds (or subtracts) an object of type `extent<N>` from this extent to form a new extent. The `result extent<N>` is such that for a given operator  $\oplus$ ,  

$$\text{result}[i] = \text{this}[i] \oplus \text{ext}[i]$$

**Parameters:**

|                  |                                                                        |
|------------------|------------------------------------------------------------------------|
| <code>ext</code> | The right-hand <code>extent&lt;N&gt;</code> to be added or subtracted. |
|------------------|------------------------------------------------------------------------|

1257

```
extent<N> operator+(const index<N>& idx) restrict(amp,cpu);
extent<N> operator-(const index<N>& idx) restrict(amp,cpu);
extent<N>& operator+=(const index<N>& idx) restrict(amp,cpu);
extent<N>& operator-=(const index<N>& idx) restrict(amp,cpu);
```

Adds (or subtracts) an object of type `index<N>` from this extent to form a new extent. The `result extent<N>` is such that for a given operator  $\oplus$ ,  

$$\text{result}[i] = \text{this}[i] \oplus \text{idx}[i]$$

**Parameters:**

|                  |                                                                       |
|------------------|-----------------------------------------------------------------------|
| <code>idx</code> | The right-hand <code>index&lt;N&gt;</code> to be added or subtracted. |
|------------------|-----------------------------------------------------------------------|

1258

1259

```
template <int N>
    friend extent<N> operator+(const extent<N>& ext, int value) restrict(amp,cpu);
template <int N>
    friend extent<N> operator+(int value, const extent<N>& ext) restrict(amp,cpu);
```

```

template <int N>
friend extent<N> operator-(const extent<N>& ext, int value) restrict(amp,cpu);
template <int N>
friend extent<N> operator-(int value, const extent<N>& ext) restrict(amp,cpu);
template <int N>
friend extent<N> operator*(const extent<N>& ext, int value) restrict(amp,cpu);
template <int N>
friend extent<N> operator*(int value, const extent<N>& ext) restrict(amp,cpu);
template <int N>
friend extent<N> operator/(const extent<N>& ext, int value) restrict(amp,cpu);
template <int N>
friend extent<N> operator/(int value, const extent<N>& ext) restrict(amp,cpu);
template <int N>
friend extent<N> operator%(const extent<N>& ext, int value) restrict(amp,cpu);
template <int N>
friend extent<N> operator%(int value, const extent<N>& ext) restrict(amp,cpu);

```

Binary arithmetic operations that produce a new `extent<N>` that is the result of performing the corresponding binary arithmetic operation on the elements of the extent operands. The `result extent<N>` is such that for a given operator  $\oplus$ ,

$$\text{result}[i] = \text{ext}[i] \oplus \text{value}$$

or

$$\text{result}[i] = \text{value} \oplus \text{ext}[i]$$

for every  $i$  from 0 to  $N-1$ .

**Parameters:**

|                    |                                          |
|--------------------|------------------------------------------|
| <code>ext</code>   | The <code>extent&lt;N&gt;</code> operand |
| <code>value</code> | The integer operand                      |

1260

```

extent& operator+=(int value) restrict(amp,cpu);
extent& operator-=(int value) restrict(amp,cpu);
extent& operator*=(int value) restrict(amp,cpu);
extent& operator/=(int value) restrict(amp,cpu);
extent& operator%=(int value) restrict(amp,cpu);

```

For a given operator  $\oplus$ , produces the same effect as  
 $(\text{*this}) = (\text{*this}) \oplus \text{value}$

The return value is “`*this`”.

**Parameters:**

|                    |                                                              |
|--------------------|--------------------------------------------------------------|
| <code>Value</code> | The right-hand <code>int</code> of the arithmetic operation. |
|--------------------|--------------------------------------------------------------|

1261

1262

```

extent& operator++() restrict(amp,cpu);
extent operator++(int) restrict(amp,cpu);
extent& operator--() restrict(amp,cpu);
extent operator--(int) restrict(amp,cpu);

```

For a given operator  $\oplus$ , produces the same effect as  
 $(\text{*this}) = (\text{*this}) \oplus 1$

For prefix increment and decrement, the return value is “`*this`”. Otherwise a new `extent<N>` is returned.

1263

1264

## 4.3 tiled\_extent<D0,D1,D2>

1265

1266

A `tiled_extent` is an extent of 1 to 3 dimensions which also subdivides the index space into 1-, 2-, or 3-dimensional tiles. It has three specialized forms: `tiled_extent<D0>`, `tiled_extent<D0,D1>`, and `tiled_extent<D0,D1,D2>`, where `D0-2` specify the positive length of the tile along each dimension, with `D0` being the most-significant dimension and `D2` being the least-significant. Partial template specializations are provided to represent 2-D and 1-D tiled extents.

1271

1272 A *tiled\_extent* can be formed from an extent by calling *extent<N>::tile<D0,D1,D2>()* or one of the other two specializations  
 1273 of *extent<N>::tile()*.

1275 A *tiled\_extent* inherits from *extent*, thus all public members of *extent* are available on *tiled\_extent*.  
 1276

1277 **4.3.1 Synopsis**

```

1278
1279
1280 template <int D0, int D1=0, int D2=0>
1281 class tiled_extent : public extent<3>
1282 {
1283     public:
1284         tiled_extent() restrict(amp,cpu);
1285         tiled_extent(const tiled_extent& other) restrict(amp,cpu);
1286         tiled_extent(const extent<3>& extent) restrict(amp,cpu);
1287
1288         tiled_extent& operator=(const tiled_extent& other) restrict(amp,cpu);
1289
1290         tiled_extent pad() const restrict(amp,cpu);
1291         tiled_extent truncate() const restrict(amp,cpu);
1292
1293     // Microsoft-specific:
1294     __declspec(property(get=get_tile_extent)) extent<3> tile_extent;
1295
1296     extent<3> get_tile_extent() const restrict(amp,cpu);
1297
1298     static const int tile_dim0 = D0;
1299     static const int tile_dim1 = D1;
1300     static const int tile_dim2 = D2;
1301
1302     friend bool operator==(const tiled_extent& lhs,
1303                             const tiled_extent& rhs) restrict(amp,cpu);
1303     friend bool operator!=(const tiled_extent& lhs,
1304                             const tiled_extent& rhs) restrict(amp,cpu);
1305 };
1306
1307
1308 template <int D0, int D1>
1309 class tiled_extent<D0,D1,0> : public extent<2>
1310 {
1311     public:
1312         tiled_extent() restrict(amp,cpu);
1313         tiled_extent(const tiled_extent& other) restrict(amp,cpu);
1314         tiled_extent(const extent<2>& extent) restrict(amp,cpu);
1315
1316         tiled_extent& operator=(const tiled_extent& other) restrict(amp,cpu);
1317
1318         tiled_extent pad() const restrict(amp,cpu);
1319         tiled_extent truncate() const restrict(amp,cpu);
1320
1321     // Microsoft-specific:
1322     __declspec(property(get=get_tile_extent)) extent<2> tile_extent;
1323
1324     extent<2> get_tile_extent() const restrict(amp,cpu);

```

```

1324
1325     static const int tile_dim0 = D0;
1326     static const int tile_dim1 = D1;
1327
1328     friend bool operator==(const tiled_extent& lhs,
1329                             const tiled_extent& rhs) restrict(amp,cpu);
1330     friend bool operator!=(const tiled_extent& lhs,
1331                             const tiled_extent& rhs) restrict(amp,cpu);
1332 };
1333
1334 template <int D0>
1335 class tiled_extent<D0,0,0> : public extent<1>
1336 {
1337 public:
1338     tiled_extent() restrict(amp,cpu);
1339     tiled_extent(const tiled_extent& other) restrict(amp,cpu);
1340     tiled_extent(const extent<1>& extent) restrict(amp,cpu);
1341
1342     tiled_extent& operator=(const tiled_extent& other) restrict(amp,cpu);
1343
1344     tiled_extent pad() const restrict(amp,cpu);
1345     tiled_extent truncate() const restrict(amp,cpu);
1346
1347     // Microsoft-specific:
1348     __declspec(property(get=get_tile_extent)) extent<1> tile_extent;
1349
1350     extent<1> get_tile_extent() const restrict(amp,cpu);
1351
1352     static const int tile_dim0 = D0;
1353
1354     friend bool operator==(const tiled_extent& lhs,
1355                             const tiled_extent& rhs) restrict(amp,cpu);
1356     friend bool operator!=(const tiled_extent& lhs,
1357                             const tiled_extent& rhs) restrict(amp,cpu);
1358 };
1359
1360
template <int D0, int D1=0, int D2=0> class tiled_extent;
template <int D0, int D1>         class tiled_extent<D0,D1,0>;
template <int D0>                 class tiled_extent<D0,0,0>;

```

Represents an extent subdivided into 1-, 2-, or 3-dimensional tiles.

#### Template Arguments

*D0, D1, D2*

The length of the tile in each specified dimension, where D0 is the most-significant dimension and D2 is the least-significant.

1361

### 4.3.2 Constructors

1362

|                                                |  |
|------------------------------------------------|--|
| <code>tiled_extent() restrict(amp,cpu);</code> |  |
|------------------------------------------------|--|

Default constructor. The origin and extent is default-constructed and thus zero.

|                    |  |
|--------------------|--|
| <b>Parameters:</b> |  |
|--------------------|--|

None.

1363

|                                                                             |  |
|-----------------------------------------------------------------------------|--|
| <code>tiled_extent(const tiled_extent&amp; other) restrict(amp,cpu);</code> |  |
|-----------------------------------------------------------------------------|--|

Copy constructor. Constructs a new `tiled_extent` from the supplied argument "other".

|      |                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                    |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      | <b>Parameters:</b><br><i>other</i>                                                                                                                                                                                                                                                                 | An object of type <code>tiled_extent</code> from which to initialize this new extent.                                                                                                                                                                                                                                                                                              |
| 1365 | <code>tiled_extent(const extent&lt;N&gt;&amp; extent) restrict(amp,cpu);</code><br>Constructs a <code>tiled_extent&lt;N&gt;</code> with the extent "extent".<br>Notice that this constructor allows implicit conversions from <code>extent&lt;N&gt;</code> to <code>tiled_extent&lt;N&gt;</code> . |                                                                                                                                                                                                                                                                                                                                                                                    |
|      | <b>Parameters:</b><br><i>extent</i>                                                                                                                                                                                                                                                                | The extent of this <code>tiled_extent</code>                                                                                                                                                                                                                                                                                                                                       |
| 1366 |                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                    |
| 1367 | <b>4.3.3 Members</b>                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                    |
| 1368 | <code>tiled_extent&amp; operator=(const tiled_extent&amp; other) restrict(amp,cpu);</code><br>Assigns the component values of "other" to this <code>tiled_extent&lt;N&gt;</code> object.                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                    |
|      | <b>Parameters:</b><br>Other                                                                                                                                                                                                                                                                        | An object of type <code>tiled_extent&lt;N&gt;</code> from which to copy into this.                                                                                                                                                                                                                                                                                                 |
|      | <b>Return Value:</b>                                                                                                                                                                                                                                                                               | Returns <code>*this</code> .                                                                                                                                                                                                                                                                                                                                                       |
| 1369 | <code>tiled_extent pad() const restrict(amp,cpu);</code><br>Returns a new <code>tiled_extent</code> with the extents adjusted <u>up</u> to be evenly divisible by the tile dimensions. The origin of the new <code>tiled_extent</code> is the same as the origin of this one.                      |                                                                                                                                                                                                                                                                                                                                                                                    |
| 1370 | <code>tiled_extent truncate() const restrict(amp,cpu);</code><br>Returns a new <code>tiled_extent</code> with the extents adjusted <u>down</u> to be evenly divisible by the tile dimensions. The origin of the new <code>tiled_extent</code> is the same as the origin of this one.               |                                                                                                                                                                                                                                                                                                                                                                                    |
| 1371 | <code>_declspec(property(get=get_tile_extent)) extent&lt;N&gt; tile_extent;</code><br><code>extent&lt;N&gt; get_tile_extent() const restrict(amp,cpu);</code>                                                                                                                                      | Returns an instance of an <code>extent&lt;N&gt;</code> that captures the values of the <code>tiled_extent</code> template arguments D0, D1, and D2.<br>For example:<br><br><code>tiled_extent&lt;64,16,4&gt; tg;<br/>extent&lt;3&gt; myTileExtent = tg.tile_extent;<br/>assert(myTileExtent[0] == 64);<br/>assert(myTileExtent[1] == 16);<br/>assert(myTileExtent[2] == 4);</code> |
| 1372 | <code>static const int tile_dim0;</code><br><code>static const int tile_dim1;</code><br><code>static const int tile_dim2;</code>                                                                                                                                                                   | These constants allow access to the template arguments of <code>tiled_extent</code> .                                                                                                                                                                                                                                                                                              |
| 1373 |                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                    |
| 1374 | <b>4.3.4 Operators</b>                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                    |
| 1375 | <code>friend bool operator==(const tiled_extent&amp; lhs,<br/>                           const tiled_extent&amp; rhs) restrict(amp,cpu);</code><br><code>friend bool operator!=(const tiled_extent&amp; lhs,<br/>                           const tiled_extent&amp; rhs) restrict(amp,cpu);</code> | Compares two objects of <code>tiled_extent&lt;N&gt;</code> .<br><br>The expression<br><i>lhs</i> $\oplus$ <i>rhs</i><br>is true if <i>lhs.extent</i> $\oplus$ <i>rhs.extent</i> and <i>lhs.origin</i> $\oplus$ <i>rhs.origin</i> .                                                                                                                                                 |
|      | <b>Parameters:</b><br><i>lhs</i>                                                                                                                                                                                                                                                                   | The left-hand <code>tiled_extent</code> to be compared.                                                                                                                                                                                                                                                                                                                            |

|            |                                                          |
|------------|----------------------------------------------------------|
| <i>rhs</i> | The right-hand <code>tiled_extent</code> to be compared. |
|------------|----------------------------------------------------------|

1376  
13771378 **4.4 tiled\_index<D0,D1,D2>**1379  
1380  
1381  
1382  
1383

A `tiled_index` is a set of indices of 1 to 3 dimensions which have been subdivided into 1-, 2-, or 3-dimensional tiles in a `tiled_extent`. It has three specialized forms: `tiled_index<D0>`, `tiled_index<D0,D1>`, and `tiled_index<D0,D1,D2>`, where  $D_{0..2}$  specify the length of the tile along each dimension, with  $D_0$  being the most-significant dimension and  $D_2$  being the least-significant. Partial template specializations are provided to represent 2-D and 1-D tiled indices.

1384  
1385

A `tiled_index` is implicitly convertible to an `index<N>`, where the implicit index represents the global index.

1386  
1387  
1388

A `tiled_index` contains 4 member indices which are related to each other mathematically and help the user to pinpoint a global index to an index within a tiled space.

1389  
1390  
1391

A `tiled_index` contains a global index into an extent space. The other indices obey the following relations:

1392  
1393  
1394  
1395

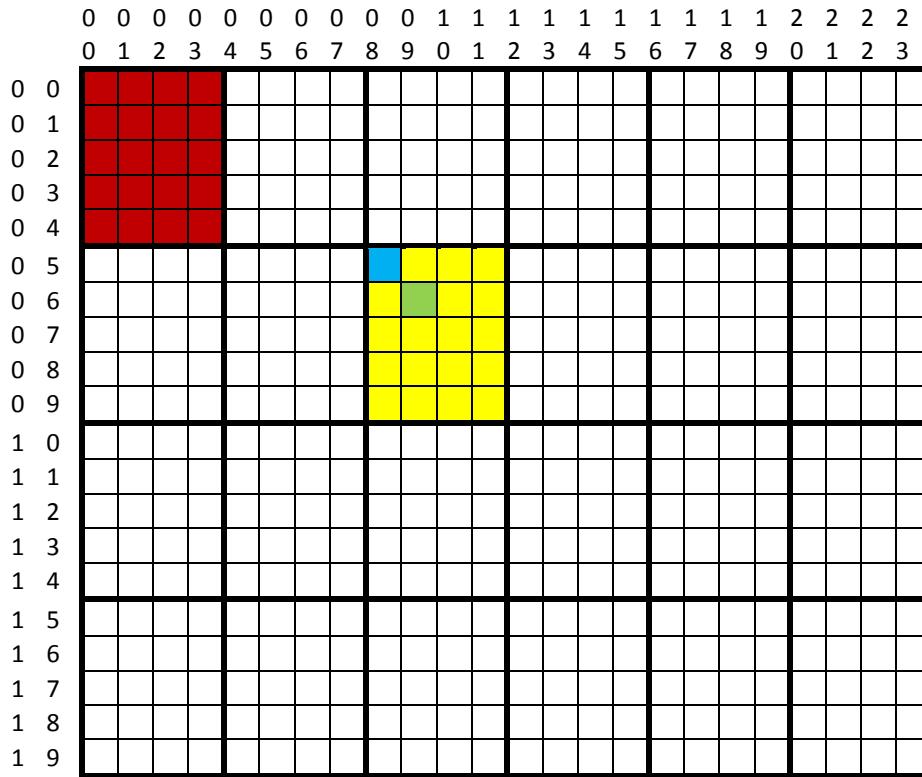
$$\begin{aligned} \text{.local} &\equiv \text{.global \% } (D_0, D_1, D_2) \\ \text{.tile} &\equiv \text{.global / } (D_0, D_1, D_2) \\ \text{.tile\_origin} &\equiv \text{.global - .local} \end{aligned}$$

1396  
1397

This is shown visually in the following example:

1398  
1399  
1400

```
parallel_for_each(extent<2>(20,24).tile<5,4>(),
    [&](tiled_index<5,4> ti) { /* ... */ });
```



1401

- 1402     1. Each cell in the diagram represents one thread which is scheduled by the `parallel_for_each` call. We see that, as  
 1403       with the non-tiled `parallel_for_each`, the number of threads scheduled is given by the extent parameter to the  
 1404       `parallel_for_each` call.  
 1405     2. Using vector notation, we see that the total number of tiles scheduled is  $<20,24> / <5,4> = <4,6>$ , which we see in  
 1406       the above diagram as 4 tiles along the vertical axis, and 6 tiles along the horizontal axis.  
 1407     3. The tile in red is tile number  $<0,0>$ . The tile in yellow is tile number  $<1,2>$ .  
 1408     4. The thread in blue:  
 1409       a. has a global id of  $<5,8>$   
 1410       b. Has a local id  $<0,0>$  within its tile. i.e., it lies on the origin of the tile.  
 1411     5. The thread in green:  
 1412       a. has a global id of  $<6,9>$   
 1413       b. has a local id of  $<1,1>$  within its tile  
 1414       c. The blue thread (number  $<5,8>$ ) is the green thread's tile origin.

#### 1416     4.4.1 Synopsis

```
1417
1418 template <int D0, int D1=0, int D2=0>
1419 class tiled_index
1420 {
1421 public:
1422     static const int rank = 3;
1423
1424     const index<3> global;
1425     const index<3> local;
1426     const index<3> tile;
1427     const index<3> tile_origin;
1428     const tile_barrier barrier;
1429
1430     tiled_index(const index<3>& global,
1431                 const index<3>& local,
1432                 const index<3>& tile,
1433                 const index<3>& tile_origin,
1434                 const tile_barrier& barrier) restrict(amp,cpu);
1435     tiled_index(const tiled_index& other) restrict(amp,cpu);
1436
1437     operator const index<3>() const restrict(amp,cpu);
1438
1439 // Microsoft-specific:
1440 __declspec(property(get=get_tile_extent)) extent<3> tile_extent;
1441
1442     extent<3> get_tile_extent() const restrict(amp,cpu);
1443
1444     static const int tile_dim0 = D0;
1445     static const int tile_dim1 = D1;
1446     static const int tile_dim2 = D2;
1447 };
1448
1449 template <int D0, int D1>
1450 class tiled_index<D0,D1,0>
1451 {
1452 public:
1453     static const int rank = 2;
1454
1455     const index<2> global;
```

```

1455     const index<2> local;
1456     const index<2> tile;
1457     const index<2> tile_origin;
1458     const tile_barrier barrier;
1459
1460     tiled_index(const index<2>& global,
1461                 const index<2>& local,
1462                 const index<2>& tile,
1463                 const index<2>& tile_origin,
1464                 const tile_barrier& barrier) restrict(amp,cpu);
1465     tiled_index(const tiled_index& other) restrict(amp,cpu);
1466
1467     operator const index<2>() const restrict(amp,cpu);
1468
1469 // Microsoft-specific:
1470 __declspec(property(get=get_tile_extent)) extent<2> tile_extent;
1471
1472     extent<2> get_tile_extent() const restrict(amp,cpu);
1473
1474     static const int tile_dim0 = D0;
1475     static const int tile_dim1 = D1;
1476 };
1477
1478 template <int D0>
1479 class tiled_index<D0,0,0>
1480 {
1481 public:
1482     static const int rank = 1;
1483
1484     const index<1> global;
1485     const index<1> local;
1486     const index<1> tile;
1487     const index<1> tile_origin;
1488     const tile_barrier barrier;
1489
1490     tiled_index(const index<1>& global,
1491                 const index<1>& local,
1492                 const index<1>& tile,
1493                 const index<1>& tile_origin,
1494                 const tile_barrier& barrier) restrict(amp,cpu);
1495     tiled_index(const tiled_index& other) restrict(amp,cpu);
1496
1497     operator const index<1>() const restrict(amp,cpu);
1498
1499 // Microsoft-specific:
1500 __declspec(property(get=get_tile_extent)) extent<1> tile_extent;
1501
1502     extent<1> get_tile_extent() const restrict(amp,cpu);
1503     static const int tile_dim0 = D0;
1504 };
1505
template <int D0, int D1=0, int D2=0> class tiled_index;
template <int D0, int D1>           class tiled_index<D0,D1,0>;

```

|                                                                                     |                                                                                                                                 |
|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <code>template &lt;int D0 &gt;</code>                                               | <code>class tiled_index&lt;D0,0,0&gt;;</code>                                                                                   |
| Represents a set of related indices subdivided into 1-, 2-, or 3-dimensional tiles. |                                                                                                                                 |
| <b>Template Arguments</b>                                                           |                                                                                                                                 |
| <code>D0, D1, D2</code>                                                             | The length of the tile in each specified dimension, where D0 is the most-significant dimension and D2 is the least-significant. |

1506

|                                         |                                                                                                                                                         |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>static const int rank = N;</code> | A static member of <code>tiled_index</code> that contains the rank of this tiled extent, and is either 1, 2, or 3 depending on the specialization used. |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|

1507

#### 1508 4.4.2 Constructors

1509

1510 The `tiled_index` class has no default constructor.

1511

|                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>tiled_index(const index&lt;N&gt;&amp; global,</code><br><code>const index&lt;N&gt;&amp; local,</code><br><code>const index&lt;N&gt;&amp; tile,</code><br><code>const index&lt;N&gt;&amp; tile_origin,</code><br><code>const tile_barrier&amp; barrier) restrict(amp,cpu);</code>                                                                                                                      | Construct a new <code>tiled_index</code> out of the constituent indices.                                                |
| Note that it is permissible to create a <code>tiled_index</code> instance for which the geometric identities which are guaranteed for system-created tiled indices, which are passed as a kernel parameter to the tiled overloads of <code>parallel_for_each</code> , do not hold. In such cases, it is up to the application to assign application-specific meaning to the member indices of the instance. |                                                                                                                         |
| <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                         |
| <code>global</code>                                                                                                                                                                                                                                                                                                                                                                                         | An object of type <code>index&lt;N&gt;</code> which is taken to be the global index of this tile.                       |
| <code>local</code>                                                                                                                                                                                                                                                                                                                                                                                          | An object of type <code>index&lt;N&gt;</code> which is taken to be the local index within this tile.                    |
| <code>tile</code>                                                                                                                                                                                                                                                                                                                                                                                           | An object of type <code>index&lt;N&gt;</code> which is taken to be the coordinates of the current tile.                 |
| <code>tile_origin</code>                                                                                                                                                                                                                                                                                                                                                                                    | An object of type <code>index&lt;N&gt;</code> which is taken to be the global index of the top-left corner of the tile. |
| <code>barrier</code>                                                                                                                                                                                                                                                                                                                                                                                        | An object of type <code>tile_barrier</code> .                                                                           |

1512

|                                                                           |                                                                                                 |
|---------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <code>tiled_index(const tiled_index&amp; other) restrict(amp,cpu);</code> | Copy constructor. Constructs a new <code>tiled_index</code> from the supplied argument "other". |
| <b>Parameters:</b>                                                        |                                                                                                 |
| <code>other</code>                                                        | An object of type <code>tiled_index</code> from which to initialize this.                       |

1513

#### 1514 4.4.3 Members

1515

|                                           |                                                                                |
|-------------------------------------------|--------------------------------------------------------------------------------|
| <code>const index&lt;N&gt; global;</code> | An index of rank 1, 2, or 3 that represents the global index within an extent. |
|-------------------------------------------|--------------------------------------------------------------------------------|

1516

|                                          |                                                                                                           |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| <code>const index&lt;N&gt; local;</code> | An index of rank 1, 2, or 3 that represents the relative index within the current tile of a tiled extent. |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------|

1517

|                                         |                                                                                                    |
|-----------------------------------------|----------------------------------------------------------------------------------------------------|
| <code>const index&lt;N&gt; tile;</code> | An index of rank 1, 2, or 3 that represents the coordinates of the current tile of a tiled extent. |
|-----------------------------------------|----------------------------------------------------------------------------------------------------|

1518

|                                                |                                                                                                                             |
|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>const index&lt;N&gt; tile_origin;</code> | An index of rank 1, 2, or 3 that represents the global coordinates of the origin of the current tile within a tiled extent. |
|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|

1519

```
const tile_barrier barrier;
```

An object which represents a barrier within the current tile of threads.

1520

```
operator const index<N>() const restrict(amp,cpu);
```

Implicit conversion operator that converts a tiled\_index<D0,D1,D2> into an index<N>. The implicit conversion converts to the .global index member.

1521

```
_declspec(property(get=get_tile_extent)) extent<N> tile_extent;
extent<N> get_tile_extent() const restrict(amp,cpu);
```

Returns an instance of an extent<N> that captures the values of the tiled\_index template arguments D0, D1, and D2. For example:

```
index<3> zero;
tiled_index<64,16,4> ti(index<3>(256,256,256), zero, zero, zero, mybarrier);
extent<3> myTileExtent = ti.tile_extent;
assert(myTileExtent.tile_dim0 == 64);
assert(myTileExtent.tile_dim1 == 16);
assert(myTileExtent.tile_dim2 == 4);
```

1522

```
static const int tile_dim0;
static const int tile_dim1;
static const int tile_dim2;
```

These constants allow access to the template arguments of tiled\_index.

1523

## 4.5 tile\_barrier

1525

The tile\_barrier class is a capability class that is only creatable by the system, and passed to a tiled\_parallel\_for\_each function object as part of the tiled\_index parameter. It provides member functions, such as wait, whose purpose is to synchronize execution of threads running within the thread tile.

1529

A call to wait shall not occur in non-uniform code within a thread tile. Section 3 defines uniformity and lack thereof formally.

1532

### 4.5.1 Synopsis

1533

```
class tile_barrier
{
public:
    tile_barrier(const tile_barrier& other) restrict(amp,cpu);

    void wait() const restrict(amp);
    void wait_with_all_memory_fence() const restrict(amp);
    void wait_with_global_memory_fence() const restrict(amp);
    void wait_with_tile_static_memory_fence() const restrict(amp);
};
```

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

### 4.5.2 Constructors

1546

1547

The tile\_barrier class does not have a public default constructor, only a copy-constructor.

1548

```
tile_barrier(const tile_barrier& other) restrict(amp,cpu);
```

Copy constructor. Constructs a new tile\_barrier from the supplied argument "other".

#### Parameters:

|       |                                                               |
|-------|---------------------------------------------------------------|
| Other | An object of type tile_barrier from which to initialize this. |
|-------|---------------------------------------------------------------|

1549

1550 **4.5.3 Members**

1551

1552 The tile\_barrier class does not have an assignment operator. Section 3 provides a complete description of the C++ AMP  
1553 memory model, of which class *tile\_barrier* is an important part.

1554

|                                               |
|-----------------------------------------------|
| <code>void wait() const restrict(amp);</code> |
|-----------------------------------------------|

Blocks execution of all threads in the thread tile until all threads in the tile have reached this call. Establishes a memory fence on all tile\_static and global memory operations executed by the threads in the tile such that all memory operations issued prior to hitting the barrier are visible to all other threads after the barrier has completed and none of the memory operations occurring after the barrier are executed before hitting the barrier. This is identical to *wait\_with\_all\_memory\_fence*.

1555

|                                                                     |
|---------------------------------------------------------------------|
| <code>void wait_with_all_memory_fence() const restrict(amp);</code> |
|---------------------------------------------------------------------|

Blocks execution of all threads in the thread tile until all threads in the tile have reached this call. Establishes a memory fence on all tile\_static and global memory operations executed by the threads in the tile such that all memory operations issued prior to hitting the barrier are visible to all other threads after the barrier has completed and none of the memory operations occurring after the barrier are executed before hitting the barrier. This is identical to *wait*.

1556

|                                                                        |
|------------------------------------------------------------------------|
| <code>void wait_with_global_memory_fence() const restrict(amp);</code> |
|------------------------------------------------------------------------|

Blocks execution of all threads in the thread tile until all threads in the tile have reached this call. Establishes a memory fence on global memory operations (but not tile-static memory operations) executed by the threads in the tile such that all global memory operations issued prior to hitting the barrier are visible to all other threads after the barrier has completed and none of the global memory operations occurring after the barrier are executed before hitting the barrier.

1557

|                                                                             |
|-----------------------------------------------------------------------------|
| <code>void wait_with_tile_static_memory_fence() const restrict(amp);</code> |
|-----------------------------------------------------------------------------|

Blocks execution of all threads in the thread tile until all threads in the tile have reached this call. Establishes a memory fence on tile-static memory operations (but not global memory operations) executed by the threads in the tile such that all tile\_static memory operations issued prior to hitting the barrier are visible to all other threads after the barrier has completed and none of the tile-static memory operations occurring after the barrier are executed before hitting the barrier.

1558

1559 **4.5.4 Other Memory Fences**

1560

1561 C++ AMP provides functions that serve as memory fences, which establish a happens-before relationship between memory  
1562 operations performed by threads within the same thread tile. These functions are available in the concurrency namespace.  
1563 Section 3 provides a complete description of the C++ AMP memory model.

1564

|                                                                            |
|----------------------------------------------------------------------------|
| <code>void all_memory_fence(const tile_barrier&amp;) restrict(amp);</code> |
|----------------------------------------------------------------------------|

Establishes a thread-tile scoped memory fence for both global and tile-static memory operations. This function does not imply a barrier and is therefore permitted in divergent code.

1565

|                                                                               |
|-------------------------------------------------------------------------------|
| <code>void global_memory_fence(const tile_barrier&amp;) restrict(amp);</code> |
|-------------------------------------------------------------------------------|

Establishes a thread-tile scoped memory fence for global (but not tile-static) memory operations. This function does not imply a barrier and is therefore permitted in divergent code.

1566

|                                                                                    |
|------------------------------------------------------------------------------------|
| <code>void tile_static_memory_fence(const tile_barrier&amp;) restrict(amp);</code> |
|------------------------------------------------------------------------------------|

Establishes a thread-tile scoped memory fence for tile-static (but not global) memory operations. This function does not imply a barrier and is therefore permitted in divergent code.

1567

1568 **4.6 completion\_future**

1569

1570 This class is the return type of all C++ AMP asynchronous APIs and has an interface analogous to std::shared\_future<void>. Similar to std::shared\_future, this type provides member methods such as *wait* and *get* to wait for C++ AMP asynchronous  
1571 operations to finish, and the type additionally provides a member method *then*, to specify a completion callback *functor* to

1572 be executed upon completion of a C++ AMP asynchronous operation. Further this type also contains a member method  
 1573 **`to_task`** (Microsoft specific extension) which returns a `concurrency::task` object which can be used to avail the capabilities of  
 1574 PPL tasks with C++ AMP asynchronous operations; viz. chaining continuations, cancellation etc. This essentially enables  
 1575 “wait-free” composition of C++ AMP asynchronous tasks on accelerators with CPU tasks.

#### 1576 4.6.1 Synopsis

```
1577
1578 class completion_future
1579 {
1580 public:
1581     completion_future();
1582     completion_future(const completion_future& other);
1583     completion_future(completion_future&& other);
1584     completion_future& operator=(const completion_future& other);
1585     completion_future& operator=(completion_future&& other);
1586
1587     void get() const;
1588     bool valid() const;
1589     void wait() const;
1590
1591     template <class rep, class period>
1592     std::future_status wait_for(const std::chrono::duration<rep, period>& rel_time) const;
1593     template <class clock, class duration>
1594     std::future_status wait_until(const std::chrono::time_point<clock, duration>& abs_time)
1595     const;
1596
1597     operator std::shared_future<void>() const;
1598
1599     template <typename functor>
1600     void then(const functor & func) const;
1601
1602 // Microsoft-specific:
1603     concurrency::task<void> to_task() const;
```

1604 };

#### 1605 4.6.2 Constructors

1606

|                                   |
|-----------------------------------|
| <code>completion_future();</code> |
|-----------------------------------|

Default constructor. Constructs an empty uninitialized `completion_future` object which does not refer to any asynchronous operation. Default constructed `completion_future` objects have `valid() == false`

1607

|                                                                      |
|----------------------------------------------------------------------|
| <code>completion_future (const completion_future&amp; other);</code> |
|----------------------------------------------------------------------|

Copy constructor. Constructs a new `completion_future` object that refers to the same asynchronous operation as the `other` `completion_future` object.

**Parameters:**

|                    |                                                                                 |
|--------------------|---------------------------------------------------------------------------------|
| <code>other</code> | An object of type <code>completion_future</code> from which to initialize this. |
|--------------------|---------------------------------------------------------------------------------|

1608

1609

1610

|                                                                     |
|---------------------------------------------------------------------|
| <code>completion_future (completion_future&amp;&amp; other);</code> |
|---------------------------------------------------------------------|

Move constructor. Move constructs a new `completion_future` object that refers to the same asynchronous operation as originally referred by the `other` `completion_future` object. After this constructor returns, `other.valid() == false`

**Parameters:**

|                    |                                                                                               |
|--------------------|-----------------------------------------------------------------------------------------------|
| <code>other</code> | An object of type <code>completion_future</code> which the new <code>completion_future</code> |
|--------------------|-----------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
|                    | object is to be move constructed from.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                    |                                                                                                                                                 |
| 1611               | <pre>completion_future&amp; operator=(const completion_future&amp; other);</pre> <p>Copy assignment. Copy assigns the contents of <code>other</code> to <code>this</code>. This method causes <code>this</code> to stop referring its current asynchronous operation and start referring the same asynchronous operation as <code>other</code>.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>other</code></td><td>An object of type <code>completion_future</code> which is copy assigned to <code>this</code>.</td></tr> </table>                                                                                                                                                                                                                                                                                                 | <code>other</code> | An object of type <code>completion_future</code> which is copy assigned to <code>this</code> .                                                  |
| <code>other</code> | An object of type <code>completion_future</code> which is copy assigned to <code>this</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                    |                                                                                                                                                 |
| 1612               | <pre>completion_future&amp; operator=(completion_future&amp;&amp; other);</pre> <p>Move assignment. Move assigns the contents of <code>other</code> to <code>this</code>. This method causes <code>this</code> to stop referring its current asynchronous operation and start referring the same asynchronous operation as <code>other</code>. After this method returns, <code>other.valid() == false</code></p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>other</code></td><td>An object of type <code>completion_future</code> which is move assigned to <code>this</code>.</td></tr> </table>                                                                                                                                                                                                                                   | <code>other</code> | An object of type <code>completion_future</code> which is move assigned to <code>this</code> .                                                  |
| <code>other</code> | An object of type <code>completion_future</code> which is move assigned to <code>this</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                    |                                                                                                                                                 |
| 1613               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |                                                                                                                                                 |
| 1614               | <b>4.6.3 Members</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                    |                                                                                                                                                 |
| 1615               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |                                                                                                                                                 |
| 1616               | <pre>void get() const;</pre> <p>This method is functionally identical to <code>std::shared_future&lt;void&gt;::get</code>. This method waits for the associated asynchronous operation to finish and returns only upon the completion of the asynchronous operation. If an exception was encountered during the execution of the asynchronous operation, this method throws that stored exception.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                |                    |                                                                                                                                                 |
| 1617               | <pre>bool valid() const;</pre> <p>This method is functionally identical to <code>std::shared_future&lt;void&gt;::valid</code>. This returns true if <code>this</code> <code>completion_future</code> is associated with an asynchronous operation.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                    |                                                                                                                                                 |
| 1618               | <pre>void wait() const; template &lt;class Rep, class Period&gt; std::future_status wait_for(const std::chrono::duration&lt;Rep, Period&gt;&amp; rel_time) const; template &lt;class Clock, class Duration&gt; std::future_status wait_until(const std::chrono::time_point&lt;Clock, Duration&gt;&amp; abs_time) const;</pre> <p>These methods are functionally identical to the corresponding <code>std::shared_future&lt;void&gt;</code> methods.</p> <p>The <code>wait</code> method waits for the associated asynchronous operation to finish and returns only upon completion of the associated asynchronous operation or if an exception was encountered when executing the asynchronous operation.</p> <p>The other variants are functionally identical to the <code>std::shared_future&lt;void&gt;</code> member methods with same names.</p> |                    |                                                                                                                                                 |
| 1619               | <pre>operator shared_future&lt;void&gt;() const;</pre> <p>Conversion operator to <code>std::shared_future&lt;void&gt;</code>. This method returns a <code>shared_future&lt;void&gt;</code> object corresponding to <code>this</code> <code>completion_future</code> object and refers to the same asynchronous operation.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                    |                                                                                                                                                 |
| 1620               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |                                                                                                                                                 |
| 1621               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |                                                                                                                                                 |
| 1622               | <pre>template &lt;typename Functor&gt; void then(const Functor &amp;func) const;</pre> <p>This method enables specification of a completion callback <code>func</code> which is executed upon completion of the asynchronous operation associated with <code>this</code> <code>completion_future</code> object. The completion callback <code>func</code> should have an <code>operator()</code> that is valid when invoked with non arguments, i.e., “<code>func()</code>”.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>func</code></td><td>A function object or lambda whose <code>operator()</code> is invoked upon completion of <code>this</code>'s associated asynchronous operation.</td></tr> </table>                                                                                                                    | <code>func</code>  | A function object or lambda whose <code>operator()</code> is invoked upon completion of <code>this</code> 's associated asynchronous operation. |
| <code>func</code>  | A function object or lambda whose <code>operator()</code> is invoked upon completion of <code>this</code> 's associated asynchronous operation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                    |                                                                                                                                                 |

1623

```
concurrency::task<void> to_task() const;
```

This method returns a `concurrency::task<void>` object corresponding to `this completion_future` object and refers to the same asynchronous operation. This method is a Microsoft specific extension.

1624

## 4.7 Access type

1625

The `access_type` enumeration denotes the type of access to data, in the context that it is used. This enumeration type can have one of the following values:

1626

```
enum access_type
{
    access_type_none,
    access_type_read,
    access_type_write,
    access_type_read_write = access_type_read | access_type_write,
    access_type_auto
};
```

1627

The enumerators should behave as bitwise flags.

1628

While the meaning of other values in the enumeration is self-explanatory, “`access_type_auto`” is a special value used to indicate that the choice of `access_type` (in the context it is used) is left to the implementation.

1629

## 5 Data Containers

1630

### 5.1 array<T,N>

1631

The type `array<T,N>` represents a dense and regular (not jagged) N-dimensional array which resides on a specific location such as an accelerator or the CPU. The element type of the array is `T`, which is necessarily of a type compatible with the target accelerator. While the rank of the array is determined statically and is part of the type, the extent of the array is runtime-determined, and is expressed using class `extent<N>`. A specific element of an array is selected using an instance of `index<N>`. If “`idx`” is a valid index for an array with extent “`e`”, then  $0 \leq idx[k] < e[k]$  for  $0 \leq k < N$ . Here each “`k`” is referred to as a dimension and higher-numbered dimensions are referred to as less significant.

1632

The array element type `T` shall be an *amp-compatible* whose size is a multiple of 4 bytes and shall not directly or recursively contain any concurrency containers or reference to concurrency containers.

1633

Array data is laid out contiguously in memory. Elements which differ by one in the least significant dimension are adjacent in memory. This storage layout is typically referred to as *row major* and is motivated by achieving efficient memory access given the standard mapping rules that GPUs use for assigning compute domain values to warps.

1634

Arrays are logically considered to be value types in that when an array is copied to another array, a deep copy is performed. Two arrays never point to the same data.

1635

The `array<T,N>` type is used in several distinct scenarios:

1636

- As a data container to be used in computations on an accelerator
- As a data container to hold memory on the host CPU (to be used to copy to and from other arrays)
- As a staging object to act as a fast intermediary for copying data between host and accelerator.

1637

An array can have any number of dimensions, although some functionality is specialized for `array<T,1>`, `array<T,2>`, and `array<T,3>`. The dimension defaults to 1 if the template argument is elided.

1668

## 1669 5.1.1 Synopsis

```

1670
1671 template <typename T, int N=1>
1672 class array
1673 {
1674 public:
1675     static const int rank = N;
1676     typedef T value_type;
1677
1678     explicit array(const extent<N>& extent);
1679     array(const extent<N>& extent, accelerator_view av, access_type cpu_access_type =
1680 access_type_auto);
1681     array(const extent<N>& extent, accelerator_view av, accelerator_view associated_av); // staging
1682
1683     template <typename InputIterator>
1684         array(const extent<N>& extent, InputIterator srcBegin);
1685     template <typename InputIterator>
1686         array(const extent<N>& extent, InputIterator srcBegin, InputIterator srcEnd);
1687     template <typename InputIterator>
1688         array(const extent<N>& extent, InputIterator srcBegin,
1689             accelerator_view av, accelerator_view associated_av); // staging
1690     template <typename InputIterator>
1691         array(const extent<N>& extent, InputIterator srcBegin, InputIterator srcEnd,
1692             accelerator_view av, accelerator_view associated_av); // staging
1693     template <typename InputIterator>
1694         array(const extent<N>& extent, InputIterator srcBegin, accelerator_view av,
1695             access_type cpu_access_type = access_type_auto);
1696     template <typename InputIterator>
1697         array(const extent<N>& extent, InputIterator srcBegin, InputIterator srcEnd,
1698             accelerator_view av, access_type cpu_access_type = access_type_auto);
1699
1700
1701     explicit array(const array_view<const T,N>& src);
1702     array(const array_view<const T,N>& src,
1703           accelerator_view av, accelerator_view associated_av); // staging
1704     array(const array_view<const T,N>& src, accelerator_view av,
1705           access_type cpu_access_type = access_type_auto);
1706
1707     array(const array& other);
1708     array(array&& other);
1709
1710     array& operator=(const array& other);
1711     array& operator=(array&& other);
1712
1713     array& operator=(const array_view<const T,N>& src);
1714
1715     void copy_to(array& dest) const;
1716     void copy_to(const array_view<T,N>& dest) const;
1717
1718     // Microsoft-specific:
1719     __declspec(property(get=get_extent)) extent<N> extent;
1720     __declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
1721     __declspec(property(get=get_associated_accelerator_view))
1722         accelerator_view associated_accelerator_view;
1723     __declspec(property(get=get_cpu_access_type)) access_type cpu_access_type;

```

```

1724     extent<N> get_extent() const restrict(amp,cpu);
1725     accelerator_view get_accelerator_view() const;
1726     accelerator_view get_associated_accelerator_view() const;
1727     access_type get_cpu_access_type() const;
1728
1729     T& operator[](const index<N>& idx) restrict(amp,cpu);
1730     const T& operator[](const index<N>& idx) const restrict(amp,cpu);
1731     array_view<T,N-1> operator[](int i) restrict(amp,cpu);
1732     array_view<const T,N-1> operator[](int i) const restrict(amp,cpu);
1733
1734     T& operator()(const index<N>& idx) restrict(amp,cpu);
1735     const T& operator()(const index<N>& idx) const restrict(amp,cpu);
1736     array_view<T,N-1> operator()(int i) restrict(amp,cpu);
1737     array_view<const T,N-1> operator()(int i) const restrict(amp,cpu);
1738
1739     array_view<T,N> section(const index<N>& origin, const extent<N>& ext) restrict(amp,cpu);
1740     array_view<const T,N> section(const index<N>& origin, const extent<N>& ext) const
1741     restrict(amp,cpu);
1742     array_view<T,N> section(const index<N>& origin) restrict(amp,cpu);
1743     array_view<const T,N> section(const index<N>& origin) const restrict(amp,cpu);
1744     array_view<T,N> section(const extent<N>& ext) restrict(amp,cpu);
1745     array_view<const T,N> section(const extent<N>& ext) const restrict(amp,cpu);
1746
1747     template <typename ElementType>
1748         array_view<ElementType,1> reinterpret_as() restrict(amp,cpu);
1749     template <typename ElementType>
1750         array_view<const ElementType,1> reinterpret_as() const restrict(amp,cpu);
1751
1752     template <int K>
1753         array_view<T,K> view_as(const extent<K>& viewExtent) restrict(amp,cpu);
1754     template <int K>
1755         array_view<const T,K> view_as(const extent<K>& viewExtent) const restrict(amp,cpu);
1756
1757     operator std::vector<T>() const;
1758
1759     T* data() restrict(amp,cpu);
1760     const T* data() const restrict(amp,cpu);
1761 };
1762
1763 template<typename T>
1764 class array<T,1>
1765 {
1766 public:
1767     static const int rank = 1;
1768     typedef T value_type;
1769
1770     explicit array(const extent<1>& extent);
1771     explicit array(int e0);
1772     array(const extent<1>& extent,
1773           accelerator_view av, accelerator_view associated_av); // staging
1774     array(int e0, accelerator_view av, accelerator_view associated_av); // staging
1775     array(const extent<1>& extent, accelerator_view av, access_type cpu_access_type =
1776     access_type_auto);
1777     array(int e0, accelerator_view av , access_type cpu_access_type = access_type_auto);
1778
1779     template <typename InputIterator>
1780         array(const extent<1>& extent, InputIterator srcBegin);
1781     template <typename InputIterator>

```

```

1782     array(const extent<1>& extent, InputIterator srcBegin, InputIterator srcEnd);
1783     template <typename InputIterator>
1784         array(int e0, InputIterator srcBegin);
1785     template <typename InputIterator>
1786         array(int e0, InputIterator srcBegin, InputIterator srcEnd);
1787     template <typename InputIterator>
1788         array(const extent<1>& extent, InputIterator srcBegin,
1789             accelerator_view av, accelerator_view associated_av); // staging
1790     template <typename InputIterator>
1791         array(const extent<1>& extent, InputIterator srcBegin, InputIterator srcEnd,
1792             accelerator_view av, accelerator_view associated_av); // staging
1793     template <typename InputIterator>
1794         array(int e0, InputIterator srcBegin,
1795             accelerator_view av, accelerator_view associated_av); // staging
1796     template <typename InputIterator>
1797         array(int e0, InputIterator srcBegin, InputIterator srcEnd,
1798             accelerator_view av, accelerator_view associated_av); // staging
1799     template <typename InputIterator>
1800         array(const extent<1>& extent, InputIterator srcBegin, accelerator_view av,
1801             access_type cpu_access_type = access_type_auto);
1802     template <typename InputIterator>
1803         array(const extent<1>& extent, InputIterator srcBegin, InputIterator srcEnd,
1804             accelerator_view av, access_type cpu_access_type = access_type_auto);
1805     template <typename InputIterator>
1806         array(int e0, InputIterator srcBegin, accelerator_view av,
1807             access_type cpu_access_type = access_type_auto);
1808     template <typename InputIterator>
1809         array(int e0, InputIterator srcBegin, InputIterator srcEnd, accelerator_view av,
1810             access_type cpu_access_type = access_type_auto);
1811
1812     explicit array(const array_view<const T,1>& src);
1813     array(const array_view<const T,1>& src,
1814         accelerator_view av, accelerator_view associated_av); // staging
1815     array(const array_view<const T,1>& src, accelerator_view av,
1816         access_type cpu_access_type = access_type_auto);
1817
1818     array(const array& other);
1819     array(array&& other);
1820
1821     array& operator=(const array& other);
1822     array& operator=(array&& other);
1823
1824     array& operator=(const array_view<const T,1>& src);
1825
1826     void copy_to(array& dest) const;
1827     void copy_to(const array_view<T,1>& dest) const;
1828
// Microsoft-specific:
1829     __declspec(property(get=get_extent)) extent<1> extent;
1830     __declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
1831     __declspec(property(get=get_associated_accelerator_view)) accelerator_view
1832     associated_accelerator_view;
1833     __declspec(property(get=get_cpu_access_type)) access_type cpu_access_type;
1834
extent<1> get_extent() const restrict(amp,cpu);
accelerator_view get_accelerator_view() const;
accelerator_view get_associated_accelerator_view() const;
access_type get_cpu_access_type() const;

```

```

1838
1839     T& operator[](const index<1>& idx) restrict(amp,cpu);
1840     const T& operator[](const index<1>& idx) const restrict(amp,cpu);
1841     T& operator[](int i0) restrict(amp,cpu);
1842     const T& operator[](int i0) const restrict(amp,cpu);
1843
1844     T& operator()(const index<1>& idx) restrict(amp,cpu);
1845     const T& operator()(const index<1>& idx) const restrict(amp,cpu);
1846     T& operator()(int i0) restrict(amp,cpu);
1847     const T& operator()(int i0) const restrict(amp,cpu);
1848
1849     array_view<T,1> section(const index<1>& origin, const extent<1>& ext) restrict(amp,cpu);
1850     array_view<const T,1> section(const index<1>& origin, const extent<1>& ext) const
1851     restrict(amp,cpu);
1852     array_view<T,1> section(const index<1>& origin) restrict(amp,cpu);
1853     array_view<const T,1> section(const index<1>& origin) const restrict(amp,cpu);
1854     array_view<T,1> section(const extent<1>& ext) restrict(amp,cpu);
1855     array_view<const T,1> section(const extent<1>& ext) const restrict(amp,cpu);
1856     array_view<T,1> section(int i0, int e0) restrict(amp,cpu);
1857     array_view<const T,1> section(int i0, int e0) const restrict(amp,cpu);
1858
1859     template <typename ElementType>
1860         array_view<ElementType,1> reinterpret_as() restrict(amp,cpu);
1861     template <typename ElementType>
1862         array_view<const ElementType,1> reinterpret_as() const restrict(amp,cpu);
1863
1864     template <int K>
1865         array_view<T,K> view_as(const extent<K>& viewExtent) restrict(amp,cpu);
1866     template <int K>
1867         array_view<const T,K> view_as(const extent<K>& viewExtent) const restrict(amp,cpu);
1868
1869     operator std::vector<T>() const;
1870
1871     T* data() restrict(amp,cpu);
1872     const T* data() const restrict(amp,cpu);
1873 };
1874
1875
1876 template<typename T>
1877 class array<T,2>
1878 {
1879 public:
1880     static const int rank = 2;
1881     typedef T value_type;
1882
1883     explicit array(const extent<2>& extent);
1884     array(int e0, int e1);
1885     array(const extent<2>& extent,
1886           accelerator_view av, accelerator_view associated_av); // staging
1887     array(int e0, int e1, accelerator_view av, accelerator_view associated_av); // staging
1888     array(const extent<2>& extent, accelerator_view av, access_type cpu_access_type =
1889 access_type_auto);
1890     array(int e0, int e1, accelerator_view av, access_type cpu_access_type = access_type_auto);
1891
1892     template <typename InputIterator>
1893         array(const extent<2>& extent, InputIterator srcBegin);
1894     template <typename InputIterator>
1895         array(const extent<2>& extent, InputIterator srcBegin, InputIterator srcEnd);

```

```

1896     template <typename InputIterator>
1897         array(int e0, int e1, InputIterator srcBegin);
1898     template <typename InputIterator>
1899         array(int e0, int e1, InputIterator srcBegin, InputIterator srcEnd);
1900     template <typename InputIterator>
1901         array(const extent<2>& extent, InputIterator srcBegin,
1902               accelerator_view av, accelerator_view associated_av); // staging
1903     template <typename InputIterator>
1904         array(const extent<2>& extent, InputIterator srcBegin, InputIterator srcEnd,
1905               accelerator_view av, accelerator_view associated_av); // staging
1906     template <typename InputIterator>
1907         array(int e0, int e1, InputIterator srcBegin,
1908               accelerator_view av, accelerator_view associated_av); // staging
1909     template <typename InputIterator>
1910         array(int e0, int e1, InputIterator srcBegin, InputIterator srcEnd,
1911               accelerator_view av, accelerator_view associated_av); // staging
1912     template <typename InputIterator>
1913         array(const extent<2>& extent, InputIterator srcBegin, accelerator_view av,
1914               access_type cpu_access_type = access_type_auto);
1915     template <typename InputIterator>
1916         array(const extent<2>& extent, InputIterator srcBegin, InputIterator srcEnd,
1917               accelerator_view av, access_type cpu_access_type = access_type_auto);
1918     template <typename InputIterator>
1919         array(int e0, int e1, InputIterator srcBegin, accelerator_view av,
1920               access_type cpu_access_type = access_type_auto);
1921     template <typename InputIterator>
1922         array(int e0, int e1, InputIterator srcBegin, InputIterator srcEnd, accelerator_view av,
1923               access_type cpu_access_type = access_type_auto);

1924
1925 explicit array(const array_view<const T,2>& src);
1926 array(const array_view<const T,2>& src,
1927       accelerator_view av, accelerator_view associated_av); // staging
1928 array(const array_view<const T,2>& src, accelerator_view av,
1929       access_type cpu_access_type = access_type_auto);
1930
1931 array(const array& other);
1932 array(array&& other);
1933
1934 array& operator=(const array& other);
1935 array& operator=(array&& other);
1936
1937 array& operator=(const array_view<const T,2>& src);
1938
1939 void copy_to(array& dest) const;
1940 void copy_to(const array_view<T,2>& dest) const;
1941

1942 // Microsoft-specific:
1943     __declspec(property(get=get_get_extent)) extent<2> extent;
1944     __declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
1945     __declspec(property(get=get_associated_accelerator_view)) accelerator_view
1946     associated_accelerator_view;
1947     __declspec(property(get=get_cpu_access_type)) access_type cpu_access_type;

1948 extent<2> get_extent() const restrict(amp,cpu);
1949 accelerator_view get_accelerator_view() const;
1950 accelerator_view get_associated_accelerator_view() const;
1951 access_type get_cpu_access_type() const;

```

```

1952
1953     T& operator[](const index<2>& idx) restrict(amp,cpu);
1954     const T& operator[](const index<2>& idx) const restrict(amp,cpu);
1955     array_view<T,1> operator[](int i0) restrict(amp,cpu);
1956     array_view<const T,1> operator[](int i0) const restrict(amp,cpu);
1957
1958     T& operator()(const index<2>& idx) restrict(amp,cpu);
1959     const T& operator()(const index<2>& idx) const restrict(amp,cpu);
1960     T& operator()(int i0, int i1) restrict(amp,cpu);
1961     const T& operator()(int i0, int i1) const restrict(amp,cpu);
1962
1963     array_view<T,2> section(const index<2>& origin, const extent<2>& ext) restrict(amp,cpu);
1964     array_view<const T,2> section(const index<2>& origin, const extent<2>& ext) const
1965     restrict(amp,cpu);
1966     array_view<T,2> section(const index<2>& origin) restrict(amp,cpu);
1967     array_view<const T,2> section(const index<2>& origin) const restrict(amp,cpu);
1968     array_view<T,2> section(const extent<2>& ext) restrict(amp,cpu);
1969     array_view<const T,2> section(const extent<2>& ext) const restrict(amp,cpu);
1970     array_view<T,2> section(int i0, int i1, int e0, int e1) restrict(amp,cpu);
1971     array_view<const T,2> section(int i0, int i1, int e0, int e1) const restrict(amp,cpu);
1972
1973     template <typename ElementType>
1974         array_view<ElementType,1> reinterpret_as() restrict(amp,cpu);
1975     template <typename ElementType>
1976         array_view<const ElementType,1> reinterpret_as() const restrict(amp,cpu);
1977
1978     template <int K>
1979         array_view<T,K> view_as(const extent<K>& viewExtent) restrict(amp,cpu);
1980     template <int K>
1981         array_view<const T,K> view_as(const extent<K>& viewExtent) const restrict(amp,cpu);
1982
1983     operator std::vector<T>() const;
1984
1985     T* data() restrict(amp,cpu);
1986     const T* data() const restrict(amp,cpu);
1987 };
1988
1989
1990     template<typename T>
1991     class array<T,3>
1992     {
1993     public:
1994         static const int rank = 3;
1995         typedef T value_type;
1996
1997         explicit array(const extent<3>& extent);
1998         array(int e0, int e1, int e2);
1999         array(const extent<3>& extent,
2000               accelerator_view av, accelerator_view associated_av); // staging
2001         array(int e0, int e1, int e2,
2002               accelerator_view av, accelerator_view associated_av); // staging
2003         array(const extent<3>& extent, accelerator_view av,
2004               access_type cpu_access_type = access_type_auto);
2005         array(int e0, int e1, int e2, accelerator_view av,
2006               access_type cpu_access_type = access_type_auto);
2007
2008         template <typename InputIterator>
2009         array(const extent<3>& extent, InputIterator srcBegin);

```

```

2010 template <typename InputIterator>
2011     array(const extent<3>& extent, InputIterator srcBegin, InputIterator srcEnd);
2012 template <typename InputIterator>
2013     array(int e0, int e1, int e2, InputIterator srcBegin);
2014 template <typename InputIterator>
2015     array(int e0, int e1, int e2, InputIterator srcBegin, InputIterator srcEnd);
2016 template <typename InputIterator>
2017     array(const extent<3>& extent, InputIterator srcBegin,
2018           accelerator_view av, accelerator_view associated_av); // staging
2019 template <typename InputIterator>
2020     array(const extent<3>& extent, InputIterator srcBegin, InputIterator srcEnd,
2021           accelerator_view av, accelerator_view associated_av); // staging
2022 template <typename InputIterator>
2023     array(int e0, int e1, int e2, InputIterator srcBegin,
2024           accelerator_view av, accelerator_view associated_av); // staging
2025 template <typename InputIterator>
2026     array(int e0, int e1, int e2, InputIterator srcBegin, InputIterator srcEnd,
2027           accelerator_view av, accelerator_view associated_av); // staging
2028 template <typename InputIterator>
2029     array(const extent<3>& extent, InputIterator srcBegin, accelerator_view av,
2030           access_type cpu_access_type = access_type_auto);
2031 template <typename InputIterator>
2032     array(const extent<3>& extent, InputIterator srcBegin, InputIterator srcEnd,
2033           accelerator_view av, access_type cpu_access_type = access_type_auto);
2034 template <typename InputIterator>
2035     array(int e0, int e1, int e2, InputIterator srcBegin, accelerator_view av,
2036           access_type cpu_access_type = access_type_auto);
2037 template <typename InputIterator>
2038     array(int e0, int e1, int e2, InputIterator srcBegin, InputIterator srcEnd,
2039           accelerator_view av, access_type cpu_access_type = access_type_auto);
2040
2041 explicit array(const array_view<const T,3>& src);
2042 array(const array_view<const T,3>& src,
2043       accelerator_view av, accelerator_view associated_av); // staging
2044 array(const array_view<const T,3>& src, accelerator_view av,
2045       access_type cpu_access_type = access_type_auto);
2046
2047 array(const array& other);
2048 array(array&& other);
2049
2050 array& operator=(const array& other);
2051 array& operator=(array&& other);
2052
2053 array& operator=(const array_view<const T,3>& src);
2054
2055 void copy_to(array& dest) const;
2056 void copy_to(const array_view<T,3>& dest) const;
2057
2058 // Microsoft-specific:
2059 __declspec(property(get=get_extent)) extent<3> extent;
2060 __declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
2061 __declspec(property(get=get_associated_accelerator_view))
2062 accelerator_view associated_accelerator_view;
2063 __declspec(property(get=get_cpu_access_type)) access_type cpu_access_type;
2064
2065 extent<3> get_extent() const restrict(cpu,amp);
accelerator_view get_accelerator_view() const;

```

```

2066     accelerator_view get_associated_accelerator_view() const;
2067     access_type get_cpu_access_type() const;
2068
2069     T& operator[](const index<3>& idx) restrict(amp,cpu);
2070     const T& operator[](const index<3>& idx) const restrict(amp,cpu);
2071     array_view<T,2> operator[](int i0) restrict(amp,cpu);
2072     array_view<const T,2> operator[](int i0) const restrict(amp,cpu);
2073
2074     T& operator()(const index<3>& idx) restrict(amp,cpu);
2075     const T& operator()(const index<3>& idx) const restrict(amp,cpu);
2076     T& operator()(int i0, int i1, int i2) restrict(amp,cpu);
2077     const T& operator()(int i0, int i1, int i2) const restrict(amp,cpu);
2078
2079     array_view<T,3> section(const index<3>& origin, const extent<3>& ext) restrict(amp,cpu);
2080     array_view<const T,3> section(const index<3>& origin, const extent<3>& ext) const
2081     restrict(amp,cpu);
2082     array_view<T,3> section(const index<3>& origin) restrict(amp,cpu);
2083     array_view<const T,3> section(const index<3>& origin) const restrict(amp,cpu);
2084     array_view<T,3> section(const extent<3>& ext) restrict(amp,cpu);
2085     array_view<const T,3> section(const extent<3>& ext) const restrict(amp,cpu);
2086     array_view<T,3> section(int i0, int i1, int i2,
2087                             int e0, int e1, int e2) restrict(amp,cpu);
2088     array_view<const T,3> section(int i0, int i1, int i2,
2089                             int e0, int e1, int e2) const restrict(amp,cpu);
2090
2091     template <typename ElementType>
2092         array_view<ElementType,1> reinterpret_as() restrict(amp,cpu);
2093     template <typename ElementType>
2094         array_view<const ElementType,1> reinterpret_as() const restrict(amp,cpu);
2095
2096     template <int K>
2097         array_view<T,K> view_as(const extent<K>& viewExtent) restrict(amp,cpu);
2098     template <int K>
2099         array_view<const T,K> view_as(const extent<K>& viewExtent) const restrict(amp,cpu);
2100
2101     operator std::vector<T>() const;
2102
2103     T* data() restrict(amp,cpu);
2104     const T* data() const restrict(amp,cpu);
2105 };
2106
2107 
```

**template <typename T, int N=1> class array;**

Represents an N-dimensional region of memory (with type T) located on an accelerator.

#### Template Arguments

|   |                                                           |
|---|-----------------------------------------------------------|
| T | The element type of this array                            |
| N | The dimensionality of the array, defaults to 1 if elided. |

**static const int rank = N;**

The rank of this array.

**typedef T value\_type;**

The element type of this array.

## 2111 5.1.2 Constructors

2112 There is no default constructor for `array<T,N>`. All constructors are restricted to run on the CPU only (can't be executed on  
 2113 an amp target).

2114

|                                             |  |
|---------------------------------------------|--|
| <code>array(const array&amp; other);</code> |  |
|---------------------------------------------|--|

Copy constructor. Constructs a new `array<T,N>` from the supplied argument `other`. The new array is located on the same accelerator\_view as the source array. A deep copy is performed.

**Parameters:**

|                    |                                                                                          |
|--------------------|------------------------------------------------------------------------------------------|
| <code>Other</code> | An object of type <code>array&lt;T,N&gt;</code> from which to initialize this new array. |
|--------------------|------------------------------------------------------------------------------------------|

2115

|                                            |  |
|--------------------------------------------|--|
| <code>array(array&amp;&amp; other);</code> |  |
|--------------------------------------------|--|

Move constructor. Constructs a new `array<T,N>` by moving from the supplied argument `other`.

**Parameters:**

|                    |                                                                                          |
|--------------------|------------------------------------------------------------------------------------------|
| <code>Other</code> | An object of type <code>array&lt;T,N&gt;</code> from which to initialize this new array. |
|--------------------|------------------------------------------------------------------------------------------|

2116

|                                                                 |  |
|-----------------------------------------------------------------|--|
| <code>explicit array(const extent&lt;N&gt;&amp; extent);</code> |  |
|-----------------------------------------------------------------|--|

Constructs a new array with the supplied extent, located on the default view of the default accelerator. If any components of the extent are non-positive, an exception will be thrown.

**Parameters:**

|                     |                                             |
|---------------------|---------------------------------------------|
| <code>Extent</code> | The extent in each dimension of this array. |
|---------------------|---------------------------------------------|

2117

|                                                                                                                                                                                  |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <code>explicit array&lt;T,1&gt;::array(int e0);</code><br><code>array&lt;T,2&gt;::array(int e0, int e1);</code><br><code>array&lt;T,3&gt;::array(int e0, int e1, int e2);</code> |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|

Equivalent to construction using "`array(extent<N>(e0 [, e1 [, e2 ]]))`".

**Parameters:**

|                                |                                                               |
|--------------------------------|---------------------------------------------------------------|
| <code>e0 [, e1 [, e2 ]]</code> | The component values that will form the extent of this array. |
|--------------------------------|---------------------------------------------------------------|

2118

|                                                                                                                                                                 |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <code>template &lt;typename InputIterator&gt;</code><br><code>array(const extent&lt;N&gt;&amp; extent, InputIterator srcBegin [, InputIterator srcEnd]);</code> |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|--|

Constructs a new array with the supplied extent, located on the default accelerator, initialized with the contents of a source container specified by a beginning and optional ending iterator. The source data is copied by value into this array as if by calling "`copy()`".

If the number of available container elements is less than `this->extent.size()`, undefined behavior results.

**Parameters:**

|                     |                                             |
|---------------------|---------------------------------------------|
| <code>extent</code> | The extent in each dimension of this array. |
|---------------------|---------------------------------------------|

|                       |                                                 |
|-----------------------|-------------------------------------------------|
| <code>srcBegin</code> | A beginning iterator into the source container. |
|-----------------------|-------------------------------------------------|

|                     |                                               |
|---------------------|-----------------------------------------------|
| <code>srcEnd</code> | An ending iterator into the source container. |
|---------------------|-----------------------------------------------|

2119

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <code>template &lt;typename InputIterator&gt;</code><br><code>array&lt;T,1&gt;::array(int e0, InputIterator srcBegin [, InputIterator srcEnd]);</code><br><code>template &lt;typename InputIterator&gt;</code><br><code>array&lt;T,2&gt;::array(int e0, int e1, InputIterator srcBegin [, InputIterator srcEnd]);</code><br><code>template &lt;typename InputIterator&gt;</code><br><code>array&lt;T,3&gt;::array(int e0, int e1, int e2, InputIterator srcBegin [, InputIterator srcEnd]);</code> |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|

Equivalent to construction using "`array(extent<N>(e0 [, e1 [, e2 ]]), src)`".

**Parameters:**

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------|-----------------------|-------------------------------------------------------------------------------------|--------------------------|------------------------------------------------|
|                                 | <table border="1"> <tr> <td><code>e0 [, e1 [, e2 ] ]</code></td><td>The component values that will form the extent of this array.</td></tr> <tr> <td><code>srcBegin</code></td><td>A beginning iterator into the source container.</td></tr> <tr> <td><code>srcEnd</code></td><td>An ending iterator into the source container.</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <code>e0 [, e1 [, e2 ] ]</code> | The component values that will form the extent of this array.                                                                   | <code>srcBegin</code> | A beginning iterator into the source container.                                     | <code>srcEnd</code>      | An ending iterator into the source container.  |
| <code>e0 [, e1 [, e2 ] ]</code> | The component values that will form the extent of this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| <code>srcBegin</code>           | A beginning iterator into the source container.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| <code>srcEnd</code>             | An ending iterator into the source container.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| 2120                            | <p><code>explicit array(const array_view&lt;const T,N&gt;&amp; src);</code></p> <p>Constructs a new array, located on the default view of the default accelerator, initialized with the contents of the array_view "src". The extent of this array is taken from the extent of the source array_view. The "src" is copied by value into this array as if by calling <code>"copy(src, *this)"</code> (see 5.3.2).</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>src</code></td><td>An <code>array_view</code> object from which to copy the data into this array (and also to determine the extent of this array).</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <code>src</code>                | An <code>array_view</code> object from which to copy the data into this array (and also to determine the extent of this array). |                       |                                                                                     |                          |                                                |
| <code>src</code>                | An <code>array_view</code> object from which to copy the data into this array (and also to determine the extent of this array).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| 2121                            | <p><code>array(const extent&lt;N&gt;&amp; extent, accelerator_view av, access_type cpu_access_type = access_type_auto);</code></p> <p>Constructs a new array with the supplied extent, located on the accelerator bound to the <code>accelerator_view</code> "av".</p> <p>Users can optionally specify the type of CPU access desired for "this" array thus requesting creation of an array that is accessible both on the specified <code>accelerator_view</code> "av" as well as the CPU (with the specified <code>cpu_access_type</code>). If a value other than <code>access_type_auto</code> or <code>access_type_none</code> is specified for the <code>cpu_access_type</code> parameter and the accelerator corresponding to the <code>accelerator_view</code> "av" does not support <code>cpu_shared_memory</code>, a <code>runtime_exception</code> is thrown. The <code>cpu_access_type</code> parameter has a default value of <code>access_type_auto</code> which leaves it up to the implementation to decide what type of allowed CPU access should the array be created with. The actual CPU <code>access_type</code> allowed for the created array can be queried using the <code>get_cpu_access_type</code> member method.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>extent</code></td><td>The extent in each dimension of this array.</td></tr> <tr> <td><code>av</code></td><td>An <code>accelerator_view</code> object which specifies the location of this array.</td></tr> <tr> <td><code>access_type</code></td><td>The type of CPU access desired for this array.</td></tr> </table> | <code>extent</code>             | The extent in each dimension of this array.                                                                                     | <code>av</code>       | An <code>accelerator_view</code> object which specifies the location of this array. | <code>access_type</code> | The type of CPU access desired for this array. |
| <code>extent</code>             | The extent in each dimension of this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| <code>av</code>                 | An <code>accelerator_view</code> object which specifies the location of this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| <code>access_type</code>        | The type of CPU access desired for this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| 2122                            | <p><code>array&lt;T,1&gt;::array(int e0, accelerator_view av, access_type cpu_access_type = access_type_auto);</code><br/> <code>array&lt;T,2&gt;::array(int e0, int e1, accelerator_view av, access_type cpu_access_type = access_type_auto);</code><br/> <code>array&lt;T,3&gt;::array(int e0, int e1, int e2, accelerator_view av, access_type cpu_access_type = access_type_auto);</code></p> <p>Equivalent to construction using <code>"array(extent&lt;N&gt;(e0 [, e1 [, e2 ]]), av, cpu_access_type)"</code>.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>e0 [, e1 [, e2 ] ]</code></td><td>The component values that will form the extent of this array.</td></tr> <tr> <td><code>av</code></td><td>An <code>accelerator_view</code> object which specifies the location of this array.</td></tr> <tr> <td><code>access_type</code></td><td>The type of CPU access desired for this array.</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | <code>e0 [, e1 [, e2 ] ]</code> | The component values that will form the extent of this array.                                                                   | <code>av</code>       | An <code>accelerator_view</code> object which specifies the location of this array. | <code>access_type</code> | The type of CPU access desired for this array. |
| <code>e0 [, e1 [, e2 ] ]</code> | The component values that will form the extent of this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| <code>av</code>                 | An <code>accelerator_view</code> object which specifies the location of this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| <code>access_type</code>        | The type of CPU access desired for this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |
| 2123                            | <p><code>template &lt;typename InputIterator&gt;</code><br/> <code>array(const extent&lt;N&gt;&amp; extent, InputIterator srcBegin [, InputIterator srcEnd],</code><br/> <code>accelerator_view av, access_type cpu_access_type = access_type_auto);</code></p> <p>Constructs a new array with the supplied extent, located on the accelerator bound to the <code>accelerator_view</code> "av", initialized with the contents of the source container specified by a beginning and optional ending iterator. The data is copied by value into this array as if by calling <code>"copy()"</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                 |                                                                                                                                 |                       |                                                                                     |                          |                                                |

Users can optionally specify the type of CPU access desired for "this" array thus requesting creation of an array that is accessible both on the specified accelerator\_view "av" as well as the CPU (with the specified CPU access\_type). If a value other than access\_type\_auto or access\_type\_none is specified for the cpu\_access\_type parameter and the accelerator corresponding to the accelerator\_view "av" does not support cpu\_shared\_memory, a runtime\_exception is thrown. The cpu\_access\_type parameter has a default value of access\_type\_auto which leaves it upto the implementation to decide what type of allowed CPU access should the array be created with. The actual CPU access\_type allowed for the created array can be queried using the get\_cpu\_access\_type member method.

**Parameters:**

|                    |                                                                               |
|--------------------|-------------------------------------------------------------------------------|
| <i>extent</i>      | The extent in each dimension of this array.                                   |
| <i>srcBegin</i>    | A beginning iterator into the source container.                               |
| <i>srcEnd</i>      | An ending iterator into the source container.                                 |
| <i>av</i>          | An <b>accelerator_view</b> object which specifies the location of this array. |
| <i>access_type</i> | The type of CPU access desired for this array.                                |

2124

```
array(const array_view<const T,N>& src, accelerator_view av, access_type cpu_access_type = access_type_auto);
```

Constructs a new array initialized with the contents of the array\_view "src". The extent of this array is taken from the extent of the source array\_view. The "src" is copied by value into this array as if by calling `"copy(src, *this)"` (see 5.3.2). The new array is located on the accelerator bound to the **accelerator\_view** "av".

Users can optionally specify the type of CPU access desired for "this" array thus requesting creation of an array that is accessible both on the specified accelerator\_view "av" as well as the CPU (with the specified CPU access\_type). If a value other than access\_type\_auto or access\_type\_none is specified for the cpu\_access\_type parameter and the accelerator corresponding to the accelerator\_view "av" does not support cpu\_shared\_memory, a runtime\_exception is thrown. The cpu\_access\_type parameter has a default value of access\_type\_auto which leaves it upto the implementation to decide what type of allowed CPU access should the array be created with. The actual CPU access\_type allowed for the created array can be queried using the get\_cpu\_access\_type member method.

**Parameters:**

|                    |                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>src</i>         | An <b>array_view</b> object from which to copy the data into this array (and also to determine the extent of this array). |
| <i>av</i>          | An <b>accelerator_view</b> object which specifies the location of this array                                              |
| <i>access_type</i> | The type of CPU access desired for this array.                                                                            |

2125

```
template <typename InputIterator>
array<T,1>::array(int e0, InputIterator srcBegin [, InputIterator srcEnd],
                  accelerator_view av, access_type cpu_access_type = access_type_auto);
template <typename InputIterator>
array<T,2>::array(int e0, int e1, InputIterator srcBegin [, InputIterator srcEnd],
                  accelerator_view av, access_type cpu_access_type = access_type_auto);
template <typename InputIterator>
array<T,3>::array(int e0, int e1, int e2, InputIterator srcBegin [, InputIterator srcEnd],
                  accelerator_view av, access_type cpu_access_type = access_type_auto);
```

Equivalent to construction using `"array(extent<N>(e0 [, e1 [, e2 ]]), srcBegin [, srcEnd], av, cpu_access_type)"`.

**Parameters:**

|                           |                                                               |
|---------------------------|---------------------------------------------------------------|
| <i>e0 [, e1 [, e2 ] ]</i> | The component values that will form the extent of this array. |
|---------------------------|---------------------------------------------------------------|

|                          |                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------|
| <code>srcBegin</code>    | A beginning iterator into the source container.                                     |
| <code>srcEnd</code>      | An ending iterator into the source container.                                       |
| <code>av</code>          | An <code>accelerator_view</code> object which specifies the location of this array. |
| <code>access_type</code> | The type of CPU access desired for this array.                                      |

2126

2127 

### 5.1.2.1 Staging Array Constructors

2128 Staging arrays are used as a hint to optimize repeated copies between two accelerators (in the current version practically  
 2129 this is between the CPU and an accelerator). Staging arrays are optimized for data transfers, and do not have stable user-  
 2130 space memory.

2131

2132 **Microsoft-specific:** On Windows, staging arrays are backed by DirectX staging buffers which have the correct hardware  
 2133 alignment to ensure efficient DMA transfer between the CPU and a device.

2134

2135 Staging arrays are differentiated from normal arrays by their construction with a second accelerator. Note that the  
 2136 `accelerator_view` property of a staging array returns the value of the first accelerator argument it was constructed with (`av`,  
 2137 below).

2138

2139 It is illegal to change or examine the contents of a staging array while it is involved in a transfer operation (i.e., between  
 2140 lines 17 and 22 in the following example):

```
2141
2142     1. class SimulationServer
2143     2. {
2144     3.     array<float,2> acceleratorArray;
2145     4.     array<float,2> stagingArray;
2146     5. public:
2147     6.     SimulationServer(const accelerator_view& av)
2148     7.         :acceleratorArray(extent<2>(1000,1000), av),
2149     8.         stagingArray(extent<2>(1000,1000), accelerator("cpu").default_view,
2150     9.             accelerator("gpu").default_view)
2151    10.    {
2152    11.    }
2153
2154    12.    void onCompute()
2155    13.    {
2156    14.        array<float,2>& a = acceleratorArray;
2157    15.        ApplyNetworkChanges(stagingArray.data());
2158    16.        completion_future cf1 = copy_async(stagingArray, a);
2159    17.        // Illegal to access staging array here
2160    18.        cf1.wait();
2161    19.        parallel_for_each(a.extents, [&](index<2> idx)
2162    20.        {
2163    21.            // Update a[idx] according to simulation
2164    22.        }
2165    23.        completion_future cf2 = copy_async(a, stagingArray);
2166    24.        // Illegal to access staging array here
2167    25.        cf2.wait();
2168    26.        SendToClient(stagingArray.data());
2169    27.    }
2170    28.    }
2171    29.    }
2172    30.    }
2173    31.    }
2174    32.    }
2175    33.    }
2176    34.    }
2177    35.    }
2178    36.    }
2179    37.    }
2180    38.    }
2181    39.    }
2182    40.    }
2183    41.    }
2184    42.    }
2185    43.    }
2186    44.    }
2187    45.    }
2188    46.    }
2189    47.    }
2190    48.    }
2191    49.    }
2192    50.    }
2193    51.    }
2194    52.    }
2195    53.    }
2196    54.    }
2197    55.    }
2198    56.    }
2199    57.    }
2200    58.    }
2201    59.    }
2202    60.    }
2203    61.    }
2204    62.    }
2205    63.    }
2206    64.    }
2207    65.    }
2208    66.    }
2209    67.    }
2210    68.    }
2211    69.    }
2212    70.    }
2213    71.    }
2214    72.    }
2215    73.    }
2216    74.    }
2217    75.    }
2218    76.    }
2219    77.    }
2220    78.    }
2221    79.    }
2222    80.    }
2223    81.    }
2224    82.    }
2225    83.    }
2226    84.    }
2227    85.    }
2228    86.    }
2229    87.    }
2230    88.    }
2231    89.    }
2232    90.    }
2233    91.    }
2234    92.    }
2235    93.    }
2236    94.    }
2237    95.    }
2238    96.    }
2239    97.    }
2240    98.    }
2241    99.    }
2242    100.   }
2243    101.   }
2244    102.   }
2245    103.   }
2246    104.   }
2247    105.   }
2248    106.   }
2249    107.   }
2250    108.   }
2251    109.   }
2252    110.   }
2253    111.   }
2254    112.   }
2255    113.   }
2256    114.   }
2257    115.   }
2258    116.   }
2259    117.   }
2260    118.   }
2261    119.   }
2262    120.   }
2263    121.   }
2264    122.   }
2265    123.   }
2266    124.   }
2267    125.   }
2268    126.   }
2269    127.   }
2270    128.   }
2271    129.   }
2272    130.   }
2273    131.   }
2274    132.   }
2275    133.   }
2276    134.   }
2277    135.   }
2278    136.   }
2279    137.   }
2280    138.   }
2281    139.   }
2282    140.   }
2283    141.   }
2284    142.   }
2285    143.   }
2286    144.   }
2287    145.   }
2288    146.   }
2289    147.   }
2290    148.   }
2291    149.   }
2292    150.   }
2293    151.   }
2294    152.   }
2295    153.   }
2296    154.   }
2297    155.   }
2298    156.   }
2299    157.   }
2300    158.   }
2301    159.   }
2302    160.   }
2303    161.   }
2304    162.   }
2305    163.   }
2306    164.   }
2307    165.   }
2308    166.   }
2309    167.   }
2310    168.   }
2311    169.   }
2312    170.   }
2313    171.   }
2314    172.   }
2315    173.   }
2316    174.   }
2317    175.   }
2318    176.   }
2319    177.   }
2320    178.   }
2321    179.   }
2322    180.   }
2323    181.   }
2324    182.   }
2325    183.   }
2326    184.   }
2327    185.   }
2328    186.   }
2329    187.   }
2330    188.   }
2331    189.   }
2332    190.   }
2333    191.   }
2334    192.   }
2335    193.   }
2336    194.   }
2337    195.   }
2338    196.   }
2339    197.   }
2340    198.   }
2341    199.   }
2342    200.   }
2343    201.   }
2344    202.   }
2345    203.   }
2346    204.   }
2347    205.   }
2348    206.   }
2349    207.   }
2350    208.   }
2351    209.   }
2352    210.   }
2353    211.   }
2354    212.   }
2355    213.   }
2356    214.   }
2357    215.   }
2358    216.   }
2359    217.   }
2360    218.   }
2361    219.   }
2362    220.   }
2363    221.   }
2364    222.   }
2365    223.   }
2366    224.   }
2367    225.   }
2368    226.   }
2369    227.   }
2370    228.   }
2371    229.   }
2372    230.   }
2373    231.   }
2374    232.   }
2375    233.   }
2376    234.   }
2377    235.   }
2378    236.   }
2379    237.   }
2380    238.   }
2381    239.   }
2382    240.   }
2383    241.   }
2384    242.   }
2385    243.   }
2386    244.   }
2387    245.   }
2388    246.   }
2389    247.   }
2390    248.   }
2391    249.   }
2392    250.   }
2393    251.   }
2394    252.   }
2395    253.   }
2396    254.   }
2397    255.   }
2398    256.   }
2399    257.   }
2400    258.   }
2401    259.   }
2402    260.   }
2403    261.   }
2404    262.   }
2405    263.   }
2406    264.   }
2407    265.   }
2408    266.   }
2409    267.   }
2410    268.   }
2411    269.   }
2412    270.   }
2413    271.   }
2414    272.   }
2415    273.   }
2416    274.   }
2417    275.   }
2418    276.   }
2419    277.   }
2420    278.   }
2421    279.   }
2422    280.   }
2423    281.   }
2424    282.   }
2425    283.   }
2426    284.   }
2427    285.   }
2428    286.   }
2429    287.   }
2430    288.   }
2431    289.   }
2432    290.   }
2433    291.   }
2434    292.   }
2435    293.   }
2436    294.   }
2437    295.   }
2438    296.   }
2439    297.   }
2440    298.   }
2441    299.   }
2442    300.   }
2443    301.   }
2444    302.   }
2445    303.   }
2446    304.   }
2447    305.   }
2448    306.   }
2449    307.   }
2450    308.   }
2451    309.   }
2452    310.   }
2453    311.   }
2454    312.   }
2455    313.   }
2456    314.   }
2457    315.   }
2458    316.   }
2459    317.   }
2460    318.   }
2461    319.   }
2462    320.   }
2463    321.   }
2464    322.   }
2465    323.   }
2466    324.   }
2467    325.   }
2468    326.   }
2469    327.   }
2470    328.   }
2471    329.   }
2472    330.   }
2473    331.   }
2474    332.   }
2475    333.   }
2476    334.   }
2477    335.   }
2478    336.   }
2479    337.   }
2480    338.   }
2481    339.   }
2482    340.   }
2483    341.   }
2484    342.   }
2485    343.   }
2486    344.   }
2487    345.   }
2488    346.   }
2489    347.   }
2490    348.   }
2491    349.   }
2492    350.   }
2493    351.   }
2494    352.   }
2495    353.   }
2496    354.   }
2497    355.   }
2498    356.   }
2499    357.   }
2500    358.   }
2501    359.   }
2502    360.   }
2503    361.   }
2504    362.   }
2505    363.   }
2506    364.   }
2507    365.   }
2508    366.   }
2509    367.   }
2510    368.   }
2511    369.   }
2512    370.   }
2513    371.   }
2514    372.   }
2515    373.   }
2516    374.   }
2517    375.   }
2518    376.   }
2519    377.   }
2520    378.   }
2521    379.   }
2522    380.   }
2523    381.   }
2524    382.   }
2525    383.   }
2526    384.   }
2527    385.   }
2528    386.   }
2529    387.   }
2530    388.   }
2531    389.   }
2532    390.   }
2533    391.   }
2534    392.   }
2535    393.   }
2536    394.   }
2537    395.   }
2538    396.   }
2539    397.   }
2540    398.   }
2541    399.   }
2542    400.   }
2543    401.   }
2544    402.   }
2545    403.   }
2546    404.   }
2547    405.   }
2548    406.   }
2549    407.   }
2550    408.   }
2551    409.   }
2552    410.   }
2553    411.   }
2554    412.   }
2555    413.   }
2556    414.   }
2557    415.   }
2558    416.   }
2559    417.   }
2560    418.   }
2561    419.   }
2562    420.   }
2563    421.   }
2564    422.   }
2565    423.   }
2566    424.   }
2567    425.   }
2568    426.   }
2569    427.   }
2570    428.   }
2571    429.   }
2572    430.   }
2573    431.   }
2574    432.   }
2575    433.   }
2576    434.   }
2577    435.   }
2578    436.   }
2579    437.   }
2580    438.   }
2581    439.   }
2582    440.   }
2583    441.   }
2584    442.   }
2585    443.   }
2586    444.   }
2587    445.   }
2588    446.   }
2589    447.   }
2590    448.   }
2591    449.   }
2592    450.   }
2593    451.   }
2594    452.   }
2595    453.   }
2596    454.   }
2597    455.   }
2598    456.   }
2599    457.   }
25100   458.   }
25101   459.   }
25102   460.   }
25103   461.   }
25104   462.   }
25105   463.   }
25106   464.   }
25107   465.   }
25108   466.   }
25109   467.   }
25110   468.   }
25111   469.   }
25112   470.   }
25113   471.   }
25114   472.   }
25115   473.   }
25116   474.   }
25117   475.   }
25118   476.   }
25119   477.   }
25120   478.   }
25121   479.   }
25122   480.   }
25123   481.   }
25124   482.   }
25125   483.   }
25126   484.   }
25127   485.   }
25128   486.   }
25129   487.   }
25130   488.   }
25131   489.   }
25132   490.   }
25133   491.   }
25134   492.   }
25135   493.   }
25136   494.   }
25137   495.   }
25138   496.   }
25139   497.   }
25140   498.   }
25141   499.   }
25142   500.   }
25143   501.   }
25144   502.   }
25145   503.   }
25146   504.   }
25147   505.   }
25148   506.   }
25149   507.   }
25150   508.   }
25151   509.   }
25152   510.   }
25153   511.   }
25154   512.   }
25155   513.   }
25156   514.   }
25157   515.   }
25158   516.   }
25159   517.   }
25160   518.   }
25161   519.   }
25162   520.   }
25163   521.   }
25164   522.   }
25165   523.   }
25166   524.   }
25167   525.   }
25168   526.   }
25169   527.   }
25170   528.   }
25171   529.   }
25172   530.   }
25173   531.   }
25174   532.   }
25175   533.   }
25176   534.   }
25177   535.   }
25178   536.   }
25179   537.   }
25180   538.   }
25181   539.   }
25182   540.   }
25183   541.   }
25184   542.   }
25185   543.   }
25186   544.   }
25187   545.   }
25188   546.   }
25189   547.   }
25190   548.   }
25191   549.   }
25192   550.   }
25193   551.   }
25194   552.   }
25195   553.   }
25196   554.   }
25197   555.   }
25198   556.   }
25199   557.   }
25200   558.   }
25201   559.   }
25202   560.   }
25203   561.   }
25204   562.   }
25205   563.   }
25206   564.   }
25207   565.   }
25208   566.   }
25209   567.   }
25210   568.   }
25211   569.   }
25212   570.   }
25213   571.   }
25214   572.   }
25215   573.   }
25216   574.   }
25217   575.   }
25218   576.   }
25219   577.   }
25220   578.   }
25221   579.   }
25222   580.   }
25223   581.   }
25224   582.   }
25225   583.   }
25226   584.   }
25227   585.   }
25228   586.   }
25229   587.   }
25230   588.   }
25231   589.   }
25232   590.   }
25233   591.   }
25234   592.   }
25235   593.   }
25236   594.   }
25237   595.   }
25238   596.   }
25239   597.   }
25240   598.   }
25241   599.   }
25242   600.   }
25243   601.   }
25244   602.   }
25245   603.   }
25246   604.   }
25247   605.   }
25248   606.   }
25249   607.   }
25250   608.   }
25251   609.   }
25252   610.   }
25253   611.   }
25254   612.   }
25255   613.   }
25256   614.   }
25257   615.   }
25258   616.   }
25259   617.   }
25260   618.   }
25261   619.   }
25262   620.   }
25263   621.   }
25264   622.   }
25265   623.   }
25266   624.   }
25267   625.   }
25268   626.   }
25269   627.   }
25270   628.   }
25271   629.   }
25272   630.   }
25273   631.   }
25274   632.   }
25275   633.   }
25276   634.   }
25277   635.   }
25278   636.   }
25279   637.   }
25280   638.   }
25281   639.   }
25282   640.   }
25283   641.   }
25284   642.   }
25285   643.   }
25286   644.   }
25287   645.   }
25288   646.   }
25289   647.   }
25290   648.   }
25291   649.   }
25292   650.   }
25293   651.   }
25294   652.   }
25295   653.   }
25296   654.   }
25297   655.   }
25298   656.   }
25299   657.   }
25300   658.   }
25301   659.   }
25302   660.   }
25303   661.   }
25304   662.   }
25305   663.   }
25306   664.   }
25307   665.   }
25308   666.   }
25309   667.   }
25310   668.   }
25311   669.   }
25312   670.   }
25313   671.   }
25314   672.   }
25315   673.   }
25316   674.   }
25317   675.   }
25318   676.   }
25319   677.   }
25320   678.   }
25321   679.   }
25322   680.   }
25323   681.   }
25324   682.   }
25325   683.   }
25326   684.   }
25327   685.   }
25328   686.   }
25329   687.   }
25330   688.   }
25331   689.   }
25332   690.   }
25333   691.   }
25334   692.   }
25335   693.   }
25336   694.   }
25337   695.   }
25338   696.   }
25339   697.   }
25340   698.   }
25341   699.   }
25342   700.   }
25343   701.   }
25344   702.   }
25345   703.   }
25346   704.   }
25347   705.   }
25348   706.   }
25349   707.   }
25350   708.   }
25351   709.   }
25352   710.   }
25353   711.   }
25354   712.   }
25355   713.   }
25356   714.   }
25357   715.   }
25358   716.   }
25359   717.   }
25360   718.   }
25361   719.   }
25362   720.   }
25363   721.   }
25364   722.   }
25365   723.   }
25366   724.   }
25367   725.   }
25368   726.   }
25369   727.   }
25370   728.   }
25371   729.   }
25372   730.   }
25373   731.   }
25374   732.   }
25375   733.   }
25376   734.   }
25377   735.   }
25378   736.   }
25379   737.   }
25380   738.   }
25381   739.   }
25382   740.   }
25383   741.   }
25384   742.   }
25385   743.   }
25386   744.   }
25387   745.   }
25388   746.   }
25389   747.   }
25390   748.   }
25391   749.   }
25392   750.   }
25393   751.   }
25394   752.   }
25395   753.   }
25396   754.   }
25397   755.   }
25398   756.   }
25399   757.   }
25400   758.   }
25401   759.   }
25402   760.   }
25403   761.   }
25404   762.   }
25405   763.   }
25406   764.   }
25407   765.   }
25408   766.   }
25409   767.   }
25410   768.   }
25411   769.   }
25412   770.   }
25413   771.   }
25414   772.   }
25415   773.   }
25416   774.   }
25417   775.   }
25418   776.   }
25419   777.   }
25420   778.   }
25421   779.   }
25422   780.   }
25423   781.   }
25424   782.   }
25425   783.   }
25426   784.   }
25427   785.   }
25428   786.   }
25429   787.   }
25430   788.   }
25431   789.   }
25432   790.   }
25433   791.   }
25434   792.   }
25435   793.   }
25436   794.   }
25437   795.   }
25438   796.   }
25439   797.   }
25440   798.   }
25441   799.   }
25442   800.   }
25443   801.   }
25444   802.   }
25445   803.   }
25446   804.   }
25447   805.   }
25448   806.   }
25449   807.   }
25450   808.   }
25451   809.   }
25452   810.   }
25453   811.   }
25454   812.   }
25455   813.   }
25456   814.   }
25457   815.   }
25458   816.   }
25459   817.   }
25460   818.   }
25461   819.   }
25462   820.   }
25463   821.   }
25464   822.   }
25465   823.   }
25466   824.   }
25467   825.   }
25468   826.   }
25469   827.   }
25470   828.   }
25471   829.   }
25472   830.   }
25473   831.   }
25474   832.   }
25475   833
```

|                                                                                                                                                                                                                                                                                      |                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| Constructs a staging array with the given extent, which acts as a staging area between accelerator views "av" and "associated_av". If "av" is a cpu accelerator view, this will construct a staging array which is optimized for data transfers between the CPU and "associated_av". |                                                                                          |
| <b>Parameters:</b>                                                                                                                                                                                                                                                                   |                                                                                          |
| <i>extent</i>                                                                                                                                                                                                                                                                        | The extent in each dimension of this array.                                              |
| <i>av</i>                                                                                                                                                                                                                                                                            | An <code>accelerator_view</code> object which specifies the home location of this array. |
| <i>associated_av</i>                                                                                                                                                                                                                                                                 | An <code>accelerator_view</code> object which specifies a target device accelerator.     |

2177

|                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| <pre>array&lt;T,1&gt;::array(int e0, accelerator_view av, accelerator_view associated_av); array&lt;T,2&gt;::array(int e0, int e1, accelerator_view av, accelerator_view associated_av); array&lt;T,3&gt;::array(int e0, int e1, int e2, accelerator_view av, accelerator_view associated_av);</pre> <p>Equivalent to construction using "<code>array(extent&lt;N&gt;(e0 [, e1 [, e2 ]]), av, associated_av)</code>".</p> |                                                                                          |
| <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                                        |                                                                                          |
| <i>e0 [, e1 [, e2 ] ]</i>                                                                                                                                                                                                                                                                                                                                                                                                 | The component values that will form the extent of this array.                            |
| <i>av</i>                                                                                                                                                                                                                                                                                                                                                                                                                 | An <code>accelerator_view</code> object which specifies the home location of this array. |
| <i>associated_av</i>                                                                                                                                                                                                                                                                                                                                                                                                      | An <code>accelerator_view</code> object which specifies a target device accelerator.     |

2178

|                                                                                                                                                                                                                                                                                                               |                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| <pre>template &lt;typename InputIterator&gt; array(const extent&lt;N&gt;&amp; extent, InputIterator srcBegin [, InputIterator srcEnd], accelerator_view av, accelerator_view associated_av);</pre>                                                                                                            |                                                                                          |
| Constructs a staging array with the given extent, which acts as a staging area between accelerator_views "av" (which must be the CPU accelerator) and "associated_av". The staging array will be initialized with the data specified by "src" as if by calling " <code>copy(src, *this)</code> " (see 5.3.2). |                                                                                          |
| <b>Parameters:</b>                                                                                                                                                                                                                                                                                            |                                                                                          |
| <i>extent</i>                                                                                                                                                                                                                                                                                                 | The extent in each dimension of this array.                                              |
| <i>srcBegin</i>                                                                                                                                                                                                                                                                                               | A beginning iterator into the source container.                                          |
| <i>srcEnd</i>                                                                                                                                                                                                                                                                                                 | An ending iterator into the source container.                                            |
| <i>av</i>                                                                                                                                                                                                                                                                                                     | An <code>accelerator_view</code> object which specifies the home location of this array. |
| <i>associated_av</i>                                                                                                                                                                                                                                                                                          | An <code>accelerator_view</code> object which specifies a target device accelerator.     |

2179

2180

|                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <pre>array(const array_view&lt;const T,N&gt;&amp; src, accelerator_view av, accelerator_view associated_av);</pre>                                                                                                                                                                                                                                                                                                      |                                                                                                                                 |
| Constructs a staging array initialized with the <code>array_view</code> given by "src", which acts as a staging area between accelerator_views "av" (which must be the CPU accelerator) and "associated_av". The extent of this array is taken from the extent of the source <code>array_view</code> . The staging array will be initialized from "src" as if by calling " <code>copy(src, *this)</code> " (see 5.3.2). |                                                                                                                                 |
| <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                 |
| <i>src</i>                                                                                                                                                                                                                                                                                                                                                                                                              | An <code>array_view</code> object from which to copy the data into this array (and also to determine the extent of this array). |
| <i>av</i>                                                                                                                                                                                                                                                                                                                                                                                                               | An <code>accelerator_view</code> object which specifies the home location of this array.                                        |

|                            |                                                                                      |
|----------------------------|--------------------------------------------------------------------------------------|
| <code>associated_av</code> | An <code>accelerator_view</code> object which specifies a target device accelerator. |
|----------------------------|--------------------------------------------------------------------------------------|

2181

```
template <typename InputIterator>
array<T,1>::array(int e0, InputIterator srcBegin [, InputIterator srcEnd], accelerator_view
av, accelerator_view associated_av);
template <typename InputIterator>
array<T,2>::array(int e0, int e1, InputIterator srcBegin [, InputIterator srcEnd],
                  accelerator_view av, accelerator_view associated_av);
template <typename InputIterator>
array<T,3>::array(int e0, int e1, int e2, InputIterator srcBegin [, InputIterator srcEnd],
                  accelerator_view av, accelerator_view associated_av);
```

Equivalent to construction using “`array(extent<N>(e0 [, e1 [, e2 ]]), src, av, associated_av)`”.

**Parameters:**

|                                |                                                                                          |
|--------------------------------|------------------------------------------------------------------------------------------|
| <code>e0 [, e1 [, e2 ]]</code> | The component values that will form the extent of this array.                            |
| <code>srcBegin</code>          | A beginning iterator into the source container.                                          |
| <code>srcEnd</code>            | An ending iterator into the source container.                                            |
| <code>av</code>                | An <code>accelerator_view</code> object which specifies the home location of this array. |
| <code>associated_av</code>     | An <code>accelerator_view</code> object which specifies a target device accelerator.     |

2182

2183

### 2184 5.1.3 Members

2185

```
_declspec(property(get=get_extent)) extent<N> extent;
extent<N> get_extent() const restrict(cpu,amp);
```

Access the extent that defines the shape of this array.

2186

```
_declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
accelerator_view get_accelerator_view() const;
```

This property returns the `accelerator_view` representing the location where this array has been allocated.

2187

```
_declspec(property(get=get_associated_accelerator_view)) accelerator_view
associated_accelerator_view;
accelerator_view get_associated_accelerator_view() const;
```

This property returns the `accelerator_view` representing the preferred target where this array can be copied.

2188

```
_declspec(property(get=get_cpu_access_type)) access_type cpu_access_type;
access_type get_cpu_access_type() const;
```

This property returns the CPU “`access_type`” allowed for this array.

2189

```
array& operator=(const array& other);
```

Assigns the contents of the array “`other`” to this array, using a deep copy.

**Parameters:**

|                    |                                                                                     |
|--------------------|-------------------------------------------------------------------------------------|
| <code>other</code> | An object of type <code>array&lt;T,N&gt;</code> from which to copy into this array. |
|--------------------|-------------------------------------------------------------------------------------|

**Return Value:**

|                              |
|------------------------------|
| Returns <code>*this</code> . |
|------------------------------|

2190

```
array& operator=(array&& other);
```

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      | Moves the contents of the array "other" to this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|      | <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|      | <i>other</i> An object of type <code>array&lt;T,N&gt;</code> from which to move into this array.                                                                                                                                                                                                                                                                                                                                                                                                                                |
|      | <b>Return Value:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|      | Returns <code>*this</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 2191 | <pre>array&amp; operator=(const array_view&lt;const T,N&gt;&amp; src);</pre> <p>Assigns the contents of the array_view "src", as if by calling "copy(src, *this)" (see 5.3.2).</p>                                                                                                                                                                                                                                                                                                                                              |
|      | <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|      | <i>src</i> An object of type <code>array_view&lt;T,N&gt;</code> from which to copy into this array.                                                                                                                                                                                                                                                                                                                                                                                                                             |
|      | <b>Return Value:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|      | Returns <code>*this</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 2192 | <pre>void copy_to(array&lt;T,N&gt;&amp; dest);</pre> <p>Copies the contents of this array to the array given by "dest", as if by calling "copy(*this, dest)" (see 5.3.2).</p>                                                                                                                                                                                                                                                                                                                                                   |
|      | <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|      | <i>dest</i> An object of type <code>array &lt;T,N&gt;</code> to which to copy data from this array.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 2193 | <pre>void copy_to(const array_view&lt;T,N&gt;&amp; dest);</pre> <p>Copies the contents of this array to the array_view given by "dest", as if by calling "copy(*this, dest)" (see 5.3.2).</p>                                                                                                                                                                                                                                                                                                                                   |
|      | <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|      | <i>dest</i> An object of type <code>array_view&lt;T,N&gt;</code> to which to copy data from this array.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 2194 | <pre>T* data() restrict(cpu,amp); const T* data() const restrict(cpu,amp);</pre> <p>Returns a pointer to the raw data underlying this array.</p>                                                                                                                                                                                                                                                                                                                                                                                |
|      | <b>Return Value:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|      | A (const) pointer to the first element in the linearized array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 2195 | <pre>operator std::vector&lt;T&gt;() const;</pre> <p>Implicitly converts an array to a <code>std::vector</code>, as if by "copy(*this, vector)" (see 5.3.2).</p>                                                                                                                                                                                                                                                                                                                                                                |
|      | <b>Return Value:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|      | An object of type <code>vector&lt;T&gt;</code> which contains a copy of the data contained on the array.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 2196 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 2197 | <b>5.1.4 Indexing</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 2198 | <pre>T&amp; operator[](const index&lt;N&gt;&amp; idx) restrict(cpu,amp); T&amp; operator()(const index&lt;N&gt;&amp; idx) restrict(cpu,amp);</pre> <p>Returns a reference to the element of this array that is at the location in N-dimensional space specified by "idx". Accessing array data on a location where it is not resident (e.g. from the CPU when it is resident on a GPU) results in an exception (in cpu-restricted context) or undefined behavior (in amp-restricted context).</p>                               |
|      | <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|      | <i>idx</i> An object of type <code>index&lt;N&gt;</code> from that specifies the location of the element.                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 2199 | <pre>const T&amp; operator[](const index&lt;N&gt;&amp; idx) const restrict(cpu,amp); const T&amp; operator()(const index&lt;N&gt;&amp; idx) const restrict(cpu,amp);</pre> <p>Returns a const reference to the element of this array that is at the location in N-dimensional space specified by "idx". Accessing array data on a location where it is not resident (e.g. from the CPU when it is resident on a GPU) results in an exception (in cpu-restricted context) or undefined behavior (in amp-restricted context).</p> |
|      | <b>Parameters:</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|      | <i>idx</i> An object of type <code>index&lt;N&gt;</code> from that specifies the location of the element.                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 2200 | <pre>T&amp; array&lt;T,1&gt;::operator[](int i0) restrict(cpu,amp);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                          |                                                                                 |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|---------------------------------------------------------------------------------|
|                          | <pre>T&amp; array&lt;T,1&gt;::operator()(int i0) restrict(cpu,amp); T&amp; array&lt;T,2&gt;::operator()(int i0, int i1) restrict(cpu,amp); T&amp; array&lt;T,3&gt;::operator()(int i0, int i1, int i2) restrict(cpu,amp); Equivalent to "array&lt;T,N&gt;::operator()(index&lt;N&gt;(i0 [, i1 [, i2 ]]))".</pre> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>i0 [, i1 [, i2 ]]</i></td><td>The component values that will form the index into this array.</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                     | <i>i0 [, i1 [, i2 ]]</i> | The component values that will form the index into this array.                  |
| <i>i0 [, i1 [, i2 ]]</i> | The component values that will form the index into this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                          |                                                                                 |
| 2201                     | <pre>const T&amp; array&lt;T,1&gt;::operator[](int i0) const restrict(cpu,amp); const T&amp; array&lt;T,1&gt;::operator()(int i0) const restrict(cpu,amp); const T&amp; array&lt;T,2&gt;::operator()(int i0, int i1) const restrict(cpu,amp); const T&amp; array&lt;T,3&gt;::operator()(int i0, int i1, int i2) const restrict(cpu,amp); Equivalent to "array&lt;T,N&gt;::operator()(index&lt;N&gt;(i0 [, i1 [, i2 ]])) const".</pre> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>i0 [, i1 [, i2 ]]</i></td><td>The component values that will form the index into this array.</td></tr> </table>                                                                                                                                                                                                                                                                                                                | <i>i0 [, i1 [, i2 ]]</i> | The component values that will form the index into this array.                  |
| <i>i0 [, i1 [, i2 ]]</i> | The component values that will form the index into this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                          |                                                                                 |
| 2202                     | <pre>array_view&lt;T,N-1&gt; operator[](int i0) restrict(cpu,amp); array_view&lt;const T,N-1&gt; operator[](int i0) const restrict(cpu,amp); array_view&lt;T,N-1&gt; operator()(int i0) restrict(cpu,amp); array_view&lt;const T,N-1&gt; operator()(int i0) const restrict(cpu,amp);</pre> <p>This overload is defined for array&lt;T,N&gt; where N ≥ 2.</p> <p>This mode of indexing is equivalent to projecting on the most-significant dimension. It allows C-style indexing. For example:</p> <pre>array&lt;float,4&gt; myArray(myExtents, ...);  myArray[index&lt;4&gt;(5,4,3,2)] = 7; assert(myArray[5][4][3][2] == 7);</pre> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>i0</i></td><td>An integer that is the index into the most-significant dimension of this array.</td></tr> </table> <p><b>Return Value:</b></p> <p>Returns an array_view whose dimension is one lower than that of this array.</p> | <i>i0</i>                | An integer that is the index into the most-significant dimension of this array. |
| <i>i0</i>                | An integer that is the index into the most-significant dimension of this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                          |                                                                                 |
| 2203                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                          |                                                                                 |
| 2204                     | <h2>5.1.5 View Operations</h2>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                          |                                                                                 |
| 2205                     | <pre>array_view&lt;T,N&gt; section(const index&lt;N&gt;&amp; origin, const extent&lt;N&gt;&amp; ext) restrict(cpu,amp); array_view&lt;const T,N&gt; section(const index&lt;N&gt;&amp; origin, const extent&lt;N&gt;&amp; ext) const restrict(cpu,amp);</pre> <p>See "array_view&lt;T,N&gt;::section(const index&lt;N&gt;&amp;, const extent&lt;N&gt;&amp;)" in section 5.2.5 for a description of this function.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                          |                                                                                 |
| 2206                     | <pre>array_view&lt;T,N&gt; section(const index&lt;N&gt;&amp; origin) restrict(cpu,amp); array_view&lt;const T,N&gt; section(const index&lt;N&gt;&amp; origin) const restrict(cpu,amp);</pre> <p>Equivalent to "section(idx, this-&gt;extent - idx)".</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                          |                                                                                 |
| 2207                     | <pre>array_view&lt;T,N&gt; section(const extent&lt;N&gt;&amp; ext) restrict(cpu,amp); array_view&lt;const T,N&gt; section(const extent&lt;N&gt;&amp; ext) const restrict(cpu,amp);</pre> <p>Equivalent to "section(index&lt;N&gt;(), ext)".</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                          |                                                                                 |
| 2208                     | <pre>array_view&lt;T,1&gt; array&lt;T,1&gt;::section(int i0, int e0) restrict(cpu,amp); array_view&lt;const T,1&gt; array&lt;T,1&gt;::section(int i0, int e0) const restrict(cpu,amp); array_view&lt;T,2&gt; array&lt;T,2&gt;::section(int i0, int i1, int e0, int e1) restrict(cpu,amp); array_view&lt;const T,2&gt; array&lt;T,2&gt;::section(int i0, int i1, int e0, int e1) const</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                          |                                                                                 |

```
restrict(cpu,amp);

array_view<T,3> array<T,3>::section(int i0, int i1, int i2, int e0, int e1, int e2)
restrict(cpu,amp);

array_view<const T,3> array<T,3>::section(int i0, int i1, int i2, int e0, int e1, int e2) const
restrict(cpu,amp);
```

Equivalent to "array<T,N>::section(index<N>(i0 [, i1 [, i2 ]]), extent<N>(e0 [, e1 [, e2 ]])) const".

## Parameters:

|                     |                                                               |
|---------------------|---------------------------------------------------------------|
| $i0 [, i1 [, i2 ]]$ | The component values that will form the origin of the section |
| $e0 [, e1 [, e2 ]]$ | The component values that will form the extent of the section |

2209

```
template<typename ElementType> array_view<ElementType,1> reinterpret_as() restrict(cpu,amp);
template<typename ElementType> array_view<const ElementType,1> reinterpret_as() const
restrict(cpu,amp);
```

Sometimes it is desirable to view the data of an N-dimensional array as a linear array, possibly with a (unsafe) reinterpretation of the element type. This can be achieved through the `reinterpret_as` member function. Example:

```
struct RGB { float r; float g; float b; };

array<RGB,3> a = ...;
array_view<float,1> v = a.reinterpret_as<float>();

assert(v.extent == 3*a.extent);
```

The size of the reinterpreted ElementType must evenly divide into the total size of this array.

## **Return Value:**

Returns an `array_view` from this `array<T,N>` with the element type reinterpreted from `T` to `ElementType`, and the rank reduced from `N` to 1.

2210

```
template <int K> array_view<T,K> view_as(const extent<K>& viewExtent) restrict(cpu,amp);
template <int K> array_view<const T,K> view_as(const extent<K>& viewExtent) const
restrict(cpu,amp);
```

An array of higher rank can be reshaped into an array of lower rank, or vice versa, using the `view_as` member function.  
Example:

```
array<float,1> a(100);  
  
array view<float,2> av = a.view as(extent<2>(2,50));
```

#### **Return Value:**

Returns an `array view` from this `array<T,N>` with the rank changed to K from N.

2211

## 5.2 array\_view<T,N>

2213

The `array_view<T,N>` type represents a possibly cached view into the data held in an `array<T,N>`, or a section thereof. It also provides such views over native CPU data. It exposes an indexing interface congruent to that of `array<T,N>`.

2216

Like an `array`, an `array_view` is an N-dimensional object, where N defaults to 1 if it is omitted.

2218

The array element type *T* shall be an *amp-compatible* whose size is a multiple of 4 bytes and shall not directly or recursively contain any concurrency containers or reference to concurrency containers.

2221

2222    *array\_views* may be accessed locally, where their source data lives, or remotely on a different accelerator\_view or  
 2223 coherence domain. When they are accessed remotely, views are copied and cached as necessary. Except for the effects of  
 2224 automatic caching, *array\_views* have a performance profile similar to that of arrays (small to negligible access penalty when  
 2225 accessing the data through views).

2226  
 2227 There are three remote usage scenarios:

- 2228    1. A view to a system memory pointer is passed through a *parallel\_for\_each* call to an accelerator and accessed on  
 2229       the accelerator.  
 2230    2. A view to an accelerator-residing array is passed using a *parallel\_for\_each* to another accelerator\_view and is  
 2231       accessed there.  
 2232    3. A view to an accelerator-residing array is accessed on the CPU.

2233 When any of these scenarios occur, the referenced views are implicitly copied by the system to the remote location and, if  
 2234 modified through the *array\_view*, copied back to the home location. The implementation is free to optimize copying  
 2235 changes back; may only copy changed elements, or may copy unchanged portions as well. Overlapping *array\_views* to the  
 2236 same data source are *not guaranteed to maintain aliasing between arrays/array\_views* on a remote location.

2237  
 2238 Multi-threaded access to the same data source, either directly or through views, must be synchronized by the user.

2239  
 2240 The runtime makes the following guarantees regarding caching of data inside array views.

- 2241    1. Let A be an array and V a view to the array. Then, all well-synchronized accesses to A and V in program order obey  
 2242       a serial happens-before relationship.  
 2243    2. Let A be an array and V1 and V2 be overlapping views to the array.  
 2244       • When executing on the accelerator where A has been allocated, all well-synchronized accesses through A,  
 2245       V1 and V2 are aliased through A and induce a total happens-before relationship which obeys program  
 2246       order. (No caching.)  
 2247       • Otherwise, if they are executing on different accelerators, then the behaviour of writes to V1 and V2 is  
 2248       undefined (a race).

2249 When an *array\_view* is created over a pointer in system memory, the user commits to:

- 2250    1. only changing the data accessible through the view directly through the view class, **or**  
 2251    2. adhering to the following rules when accessing the data directly (not through the view):  
 2252       a. Calling *synchronize()* before the data is accessed directly, **and**  
 2253       b. If the underlying data is modified, calling *refresh()* prior to further accessing it through the view.

2254 (Note: The underlying data of an *array\_view* is updated when the last copy of an *array\_view* having pending writes goes out  
 2255 of scope or is otherwise destructed.)

2256  
 2257 Either action will notify the *array\_view* that the underlying native memory has changed and that any accelerator-residing  
 2258 copies are now stale. If the user abides by these rules then the guarantees provided by the system for pointer-based views  
 2259 are identical to those provided to views of data-parallel arrays.

2260  
 2261 The memory allocation underlying a concurrency::array is reference counted for automatic lifetime management. The array  
 2262 and all *array\_views* created from it hold references to the allocation and the allocation lives till there exists at least one  
 2263 array or *array\_view* object that references the allocation. Thus it is legal to access the *array\_view*(s) even after the source  
 2264 concurrency::array object has been destructed.

2265  
 2266 When an *array\_view* is created over native CPU data (such as raw CPU memory, std::vector, etc.), it is the user's  
 2267 responsibility to ensure that the source data outlives all *array\_views* created over that source. Any attempt to access the  
 2268 *array\_view* contents after native CPU data has been deallocated has undefined behavior.

2269 **5.2.1 Synopsis**2270 The `array_view<T,N>` has the following specializations:

- 2271 • `array_view<T,1>`
- 2272 • `array_view<T,2>`
- 2273 • `array_view<T,3>`
- 2274 • `array_view<const T,N>`
- 2275 • `array_view<const T,1>`
- 2276 • `array_view<const T,2>`
- 2277 • `array_view<const T,3>`

2278 **5.2.1.1 array\_view<T,N>**2279 The generic `array_view<T,N>` represents a view over elements of type `T` with rank `N`. The elements are both readable and  
2280 writeable.

```

2281
2282 template <typename T, int N = 1>
2283 class array_view
2284 {
2285 public:
2286     static const int rank = N;
2287     typedef T value_type;
2288
2289     array_view(array<T,N>& src) restrict(amp,cpu);
2290     template <typename Container>
2291         array_view(const extent<N>& extent, Container& src);
2292     array_view(const extent<N>& extent, value_type* src) restrict(amp,cpu);
2293     explicit array_view(const extent<N>& extent);
2294
2295     array_view(const array_view& other) restrict(amp,cpu);
2296
2297     array_view& operator=(const array_view& other) restrict(amp,cpu);
2298
2299     void copy_to(array<T,N>& dest) const;
2300     void copy_to(const array_view& dest) const;
2301
2302     // Microsoft-specific:
2303     __declspec(property(get=get_extent)) extent<N> extent;
2304     __declspec(property(get=get_source_accelerator_view))
2305         accelerator_view source_accelerator_view;
2306
2307     extent<N> get_extent() const restrict(amp,cpu);
2308     accelerator_view get_source_accelerator_view() const;
2309
2310     T& operator[](const index<N>& idx) const restrict(amp,cpu);
2311     array_view<T,N-1> operator[](int i) const restrict(amp,cpu);
2312     T& get_ref(const index<N>& idx) const restrict(amp,cpu);
2313
2314     T& operator()(const index<N>& idx) const restrict(amp,cpu);
2315     array_view<T,N-1> operator()(int i) const restrict(amp,cpu);
2316
2317     array_view section(const index<N>& origin, const extent<N>& ext) restrict(amp,cpu);
2318     array_view section(const index<N>& origin) const restrict(amp,cpu);
2319     array_view section(const extent<N>& ext) const restrict(amp,cpu);
2320
2321     void synchronize(access_type type = access_type_read) const;

```

```

2321     completion_future synchronize_async(access_type type = access_type_read) const;
2322 
2323     void synchronize_to(const accelerator_view& av, access_type type = access_type_read) const;
2324     completion_future synchronize_to_async(const accelerator_view& av, access_type type =
2325 access_type_read) const;
2326 
2327     void refresh() const;
2328     void discard_data() const;
2329 }
2330 
2331 template <typename T>
2332 class array_view<T,1>
2333 {
2334 public:
2335     static const int rank = 1;
2336     typedef T value_type;
2337 
2338     array_view(array<T,1>& src) restrict(amp,cpu);
2339     template <typename Container>
2340         array_view(const extent<1>& extent, Container& src);
2341     template <typename Container>
2342         array_view(int e0, Container& src);
2343     array_view(const extent<1>& extent, value_type* src) restrict(amp,cpu);
2344     array_view(int e0, value_type* src) restrict(amp,cpu);
2345     explicit array_view(const extent<1>& extent);
2346     explicit array_view(int e0);
2347     template <typename Container>
2348         explicit array_view(Container& src);
2349     template <typename value_type, int Size>
2350         explicit array_view(value_type (&src) [Size]) restrict(amp,cpu);
2351 
2352     array_view(const array_view& other) restrict(amp,cpu);
2353 
2354     array_view& operator=(const array_view& other) restrict(amp,cpu);
2355 
2356     void copy_to(array<T,1>& dest) const;
2357     void copy_to(const array_view& dest) const;
2358 
2359 // Microsoft-specific:
2360     __declspec(property(get=get_extent)) extent<1> extent;
2361     __declspec(property(get=get_source_accelerator_view)) accelerator_view
2362 source_accelerator_view;
2363 
2364     extent<1> get_extent() const restrict(amp,cpu);
2365     accelerator_view get_source_accelerator_view() const;
2366 
2367     T& operator[](const index<1>& idx) const restrict(amp,cpu);
2368     T& operator[](int i) const restrict(amp,cpu);
2369     T& get_ref(const index<1>& idx) const restrict(amp,cpu);
2370 
2371     T& operator()(const index<1>& idx) const restrict(amp,cpu);
2372     T& operator()(int i) const restrict(amp,cpu);
2373 
2374     array_view section(const index<1>& origin, const extent<1>& ext) const restrict(amp,cpu);
2375     array_view section(const index<1>& origin) const restrict(amp,cpu);
2376     array_view section(const extent<1>& ext) const restrict(amp,cpu);

```

```

2377     array_view section(int i0, int e0) const restrict(amp,cpu);
2378
2379     template <typename ElementType>
2380         array_view<ElementType,1> reinterpret_as() const restrict(amp,cpu);
2381
2382     template <int K>
2383         array_view<T,K> view_as(const extent<K>& viewExtent) const restrict(amp,cpu);
2384
2385     T* data() const restrict(amp,cpu);
2386
2387     void synchronize(access_type type = access_type_read) const;
2388     completion_future synchronize_async(access_type type = access_type_read) const;
2389
2390     void synchronize_to(const accelerator_view& av, access_type type = access_type_read) const;
2391     completion_future synchronize_to_async(const accelerator_view& av, access_type type =
2392 access_type_read) const;
2393
2394     void refresh() const;
2395     void discard_data() const;
2396 };
2397
2398
2399 template <typename T>
2400 class array_view<T,2>
2401 {
2402 public:
2403     static const int rank = 2;
2404     typedef T value_type;
2405
2406     array_view(array<T,2>& src) restrict(amp,cpu);
2407     template <typename Container>
2408         array_view(const extent<2>& extent, Container& src);
2409     template <typename Container>
2410         array_view(int e0, int e1, Container& src);
2411     array_view(const extent<2>& extent, value_type* src) restrict(amp,cpu);
2412     array_view(int e0, int e1, value_type* src) restrict(amp,cpu);
2413     explicit array_view(const extent<2>& extent);
2414     explicit array_view(int e0, int e1);
2415
2416     array_view(const array_view& other) restrict(amp,cpu);
2417
2418     array_view& operator=(const array_view& other) restrict(amp,cpu);
2419
2420     void copy_to(array<T,2>& dest) const;
2421     void copy_to(const array_view& dest) const;
2422
2423 // Microsoft-specific:
2424     __declspec(property(get=get_extent)) extent<2> extent;
2425     __declspec(property(get=get_source_accelerator_view)) accelerator_view
2426 source_accelerator_view;
2427
2428     extent<2> get_extent() const restrict(amp,cpu);
2429     accelerator_view get_source_accelerator_view() const;
2430
2431     T& operator[](const index<2>& idx) const restrict(amp,cpu);
2432     array_view<T,1> operator[](int i) const restrict(amp,cpu);
2433     T& get_ref(const index<2>& idx) const restrict(amp,cpu);

```

```

2433
2434     T& operator()(const index<2>& idx) const restrict(amp,cpu);
2435     T& operator()(int i0, int i1) const restrict(amp,cpu);
2436     array_view<T,1> operator()(int i) const restrict(amp,cpu);
2437
2438     array_view section(const index<2>& origin, const extent<2>& ext) const restrict(amp,cpu);
2439     array_view section(const index<2>& origin) const restrict(amp,cpu);
2440     array_view section(const extent<2>& ext) const restrict(amp,cpu);
2441     array_view section(int i0, int i1, int e0, int e1) const restrict(amp,cpu);
2442
2443     void synchronize(access_type type = access_type_read) const;
2444     completion_future synchronize_async(access_type type = access_type_read) const;
2445
2446     void synchronize_to(const accelerator_view& av, access_type type = access_type_read) const;
2447     completion_future synchronize_to_async(const accelerator_view& av, access_type type =
2448 access_type_read) const;
2449
2450     void refresh() const;
2451     void discard_data() const;
2452 };
2453
2454 template <typename T>
2455 class array_view<T,3>
2456 {
2457 public:
2458     static const int rank = 3;
2459     typedef T value_type;
2460
2461     array_view(array<T,3>& src) restrict(amp,cpu);
2462     template <typename Container>
2463         array_view(const extent<3>& extent, Container& src);
2464     template <typename Container>
2465         array_view(int e0, int e1, int e2, Container& src);
2466     array_view(const extent<3>& extent, value_type* src) restrict(amp,cpu);
2467     array_view(int e0, int e1, int e2, value_type* src) restrict(amp,cpu);
2468     explicit array_view(const extent<3>& extent);
2469     explicit array_view(int e0, int e1, int e2);
2470
2471     array_view(const array_view& other) restrict(amp,cpu);
2472
2473     array_view& operator=(const array_view& other) restrict(amp,cpu);
2474
2475     void copy_to(array<T,3>& dest) const;
2476     void copy_to(const array_view& dest) const;
2477
2478 // Microsoft-specific:
2479     __declspec(property(get=get_extent)) extent<3> extent;
2480     __declspec(property(get=get_source_accelerator_view)) accelerator_view
2481 source_accelerator_view;
2482
2483     extent<3> get_extent() const restrict(amp,cpu);
2484     accelerator_view get_source_accelerator_view() const;
2485
2486     T& operator[](const index<3>& idx) const restrict(amp,cpu);
2487     array_view<T,2> operator[](int i) const restrict(amp,cpu);
2488     T& get_ref(const index<3>& idx) const restrict(amp,cpu);

```

```

2489 T& operator()(const index<3>& idx) const restrict(amp,cpu);
2490 T& operator()(int i0, int i1, int i2) const restrict(amp,cpu);
2491 array_view<T,2> operator()(int i) const restrict(amp,cpu);
2492
2493 array_view section(const index<3>& origin, const extent<3>& ext) const restrict(amp,cpu);
2494 array_view section(const index<3>& origin) const restrict(amp,cpu);
2495 array_view section(const extent<3>& ext) const restrict(amp,cpu);
2496 array_view section(int i0, int i1, int i2, int e0, int e1, int e2) const restrict(amp,cpu);
2497
2498 void synchronize(access_type type = access_type_read) const;
2499 completion_future synchronize_async(access_type type = access_type_read) const;
2500
2501 void synchronize_to(const accelerator_view& av, access_type type = access_type_read) const;
2502 completion_future synchronize_to_async(const accelerator_view& av, access_type type =
2503 access_type_read) const;
2504
2505 void refresh() const;
2506 void discard_data() const;
2507 };
2508

```

### 5.2.1.2 array\_view<const T,N>

The partial specialization `array_view<const T,N>` represents a view over elements of type `const T` with rank `N`. The elements are readonly. At the boundary of a call site (such as `parallel_for_each`), this form of `array_view` need only be copied to the target accelerator if it isn't already there. It will not be copied out.

```

2513
2514 template <typename T, int N=1>
2515 class array_view<const T,N>
2516 {
2517 public:
2518     static const int rank = N;
2519     typedef const T value_type;
2520
2521     array_view(const array<T,N>& src) restrict(amp,cpu);
2522     template <typename Container>
2523         array_view(const extent<N>& extent, const Container& src);
2524     array_view(const extent<N>& extent, const value_type* src) restrict(amp,cpu);
2525
2526     array_view(const array_view<T,N>& other) restrict(amp,cpu);
2527     array_view(const array_view& other) restrict(amp,cpu);
2528
2529     array_view& operator=(const array_view<T,N>& other) restrict(amp,cpu);
2530     array_view& operator=(const array_view& other) restrict(amp,cpu);
2531
2532     void copy_to(array<T,N>& dest) const;
2533     void copy_to(const array_view<T,N>& dest) const;
2534
2535     // Microsoft-specific:
2536     __declspec(property(get=get_extent)) extent<N> extent;
2537     __declspec(property(get=get_source_accelerator_view)) accelerator_view
2538     source_accelerator_view;
2539
2540     extent<N> get_extent() const restrict(amp,cpu);
2541     accelerator_view get_source_accelerator_view() const;
2542
2543     const T& operator[](const index<N>& idx) const restrict(amp,cpu);

```

```

2543     array_view<const T,N-1> operator[](int i) const restrict(amp,cpu);
2544     const T& get_ref(const index<N>& idx) const restrict(amp,cpu);
2545
2546     const T& operator()(const index<N>& idx) const restrict(amp,cpu);
2547     array_view<const T,N-1> operator()(int i) const restrict(amp,cpu);
2548
2549     array_view section(const index<N>& origin, const extent<N>& ext) const restrict(amp,cpu);
2550     array_view section(const index<N>& origin) const restrict(amp,cpu);
2551     array_view section(const extent<N>& ext) const restrict(amp,cpu);
2552
2553     void synchronize() const;
2554     completion_future synchronize_async() const;
2555
2556     void synchronize_to(const accelerator_view& av) const;
2557     completion_future synchronize_to_async(const accelerator_view& av) const;
2558
2559     void refresh() const;
2560 };
2561
2562 template <typename T>
2563 class array_view<const T,1>
2564 {
2565 public:
2566     static const int rank = 1;
2567     typedef const T value_type;
2568
2569     array_view(const array<T,1>& src) restrict(amp,cpu);
2570     template <typename Container>
2571         array_view(const extent<1>& extent, const Container& src);
2572     template <typename Container>
2573         array_view(int e0, const Container& src);
2574     array_view(const extent<1>& extent, const value_type* src) restrict(amp,cpu);
2575     array_view(int e0, const value_type* src) restrict(amp,cpu);
2576     template <typename Container>
2577         explicit array_view(const Container& src);
2578     template <typename value_type, int Size>
2579         explicit array_view(const value_type (&src) [Size]) restrict(amp,cpu);
2580
2581     array_view(const array_view<T,1>& other) restrict(amp,cpu);
2582     array_view(const array_view& other) restrict(amp,cpu);
2583
2584     array_view& operator=(const array_view<T,1>& other) restrict(amp,cpu);
2585     array_view& operator=(const array_view& other) restrict(amp,cpu);
2586
2587     void copy_to(array<T,1>& dest) const;
2588     void copy_to(const array_view<T,1>& dest) const;
2589
2590 // Microsoft-specific:
2591     __declspec(property(get=get_extent)) extent<1> extent;
2592     __declspec(property(get=get_source_accelerator_view)) accelerator_view
2593     source_accelerator_view;
2594
2595     extent<1> get_extent() const restrict(amp,cpu);
2596     accelerator_view get_source_accelerator_view() const;
2597
2598     const T& operator[](const index<1>& idx) const restrict(amp,cpu);
2599     const T& operator[](int i) const restrict(amp,cpu);

```

```

2599 const T& get_ref(const index<1>& idx) const restrict(amp,cpu);
2600
2601 const T& operator()(const index<1>& idx) const restrict(amp,cpu);
2602 const T& operator()(int i) const restrict(amp,cpu);
2603
2604 array_view section(const index<1>& origin, const extent<1>& ext) const restrict(amp,cpu);
2605 array_view section(const index<1>& origin) const restrict(amp,cpu);
2606 array_view section(const extent<1>& ext) const restrict(amp,cpu);
2607 array_view section(int i0, int e0) const restrict(amp,cpu);
2608
2609 template <typename ElementType>
2610     array_view<const ElementType,1> reinterpret_as() const restrict(amp,cpu);
2611
2612 template <int K>
2613     array_view<const T,K> view_as(const extent<K>& viewExtent) const restrict(amp,cpu);
2614
2615 const T* data() const restrict(amp,cpu);
2616
2617 void synchronize() const;
2618 completion_future synchronize_async() const;
2619
2620 void synchronize_to(const accelerator_view& av) const;
2621 completion_future synchronize_to_async(const accelerator_view& av) const;
2622
2623 void refresh() const;
2624 };
2625
2626 template <typename T>
2627 class array_view<const T,2>
2628 {
2629 public:
2630     static const int rank = 2;
2631     typedef const T value_type;
2632
2633     array_view(const array<T,2>& src) restrict(amp,cpu);
2634     template <typename Container>
2635         array_view(const extent<2>& extent, const Container& src);
2636     template <typename Container>
2637         array_view(int e0, int e1, const Container& src);
2638     array_view(const extent<2>& extent, const value_type* src) restrict(amp,cpu);
2639     array_view(int e0, int e1, const value_type* src) restrict(amp,cpu);
2640
2641     array_view(const array_view<T,2>& other) restrict(amp,cpu);
2642     array_view(const array_view& other) restrict(amp,cpu);
2643
2644     array_view& operator=(const array_view<T,2>& other) restrict(amp,cpu);
2645     array_view& operator=(const array_view& other) restrict(amp,cpu);
2646
2647     void copy_to(array<T,2>& dest) const;
2648     void copy_to(const array_view<T,2>& dest) const;
2649
2650 // Microsoft-specific:
2651     __declspec(property(get=get_extent)) extent<2> extent;
2652     __declspec(property(get=get_source_accelerator_view)) accelerator_view
2653     source_accelerator_view;
2654
2655     extent<2> get_extent() const restrict(amp,cpu);

```

```

2655 accelerator_view get_source_accelerator_view() const;
2656
2657 const T& operator[](const index<2>& idx) const restrict(amp,cpu);
2658 array_view<const T,1> operator[](int i) const restrict(amp,cpu);
2659 const T& get_ref(const index<2>& idx) const restrict(amp,cpu);
2660
2661 const T& operator()(const index<2>& idx) const restrict(amp,cpu);
2662 const T& operator()(int i0, int i1) const restrict(amp,cpu);
2663 array_view<const T,1> operator()(int i) const restrict(amp,cpu);
2664
2665 array_view section(const index<2>& origin, const extent<2>& ext) const restrict(amp,cpu);
2666 array_view section(const index<2>& origin) const restrict(amp,cpu);
2667 array_view section(const extent<2>& ext) const restrict(amp,cpu);
2668 array_view section(int i0, int i1, int e0, int e1) const restrict(amp,cpu);
2669
2670 void synchronize() const;
2671 completion_future synchronize_async() const;
2672
2673 void synchronize_to(const accelerator_view& av) const;
2674 completion_future synchronize_to_async(const accelerator_view& av) const;
2675
2676 void refresh() const;
2677 };
2678
2679 template <typename T>
2680 class array_view<const T,3>
2681 {
2682 public:
2683     static const int rank = 3;
2684     typedef const T value_type;
2685
2686     array_view(const array<T,3>& src) restrict(amp,cpu);
2687     template <typename Container>
2688         array_view(const extent<3>& extent, const Container& src);
2689     template <typename Container>
2690         array_view(int e0, int e1, int e2, const Container& src);
2691     array_view(const extent<3>& extent, const value_type* src) restrict(amp,cpu);
2692     array_view(int e0, int e1, int e2, const value_type* src) restrict(amp,cpu);
2693
2694     array_view(const array_view<T,3>& other) restrict(amp,cpu);
2695     array_view(const array_view& other) restrict(amp,cpu);
2696
2697     array_view& operator=(const array_view<T,3>& other) restrict(amp,cpu);
2698     array_view& operator=(const array_view& other) restrict(amp,cpu);
2699
2700     void copy_to(array<T,3>& dest) const;
2701     void copy_to(const array_view<T,3>& dest) const;
2702
2703 // Microsoft-specific:
2704     __declspec(property(get=get_extent)) extent<2> extent;
2705     __declspec(property(get=get_source_accelerator_view)) accelerator_view
2706 source_accelerator_view;
2707
2708     extent<3> get_extent() const restrict(amp,cpu);
2709     accelerator_view get_source_accelerator_view() const;
2710     const T& operator[](const index<3>& idx) const restrict(amp,cpu);

```

```

2711     array_view<const T,2> operator[](int i) const restrict(amp,cpu);
2712     const T& get_ref(const index<3>& idx) const restrict(amp,cpu);
2713
2714     const T& operator()(const index<3>& idx) const restrict(amp,cpu);
2715     const T& operator()(int i0, int i1, int i2) const restrict(amp,cpu);
2716     array_view<const T,2> operator()(int i) const restrict(amp,cpu);
2717
2718     array_view section(const index<3>& origin, const extent<3>& ext) const restrict(amp,cpu);
2719     array_view section(const index<3>& origin) const restrict(amp,cpu);
2720     array_view section(const extent<3>& ext) const restrict(amp,cpu);
2721     array_view section(int i0, int i1, int i2, int e0, int e1, int e2) const restrict(amp,cpu);
2722
2723     void synchronize() const;
2724     completion_future synchronize_async() const;
2725
2726     void synchronize_to(const accelerator_view& av) const;
2727     completion_future synchronize_to_async(const accelerator_view& av) const;
2728
2729     void refresh() const;
2730 };

```

## 5.2.2 Constructors

The `array_view` type cannot be default-constructed. No bounds-checking is performed when constructing `array_views`.

|                                                                                                                                                                                       |                                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <code>array_view&lt;T,N&gt;::array_view(array&lt;T,N&gt;&amp; src) restrict(amp,cpu);</code>                                                                                          | <code>array_view&lt;const T,N&gt;::array_view(const array&lt;T,N&gt;&amp; src) restrict(amp,cpu);</code> |
| Constructs an array_view which is bound to the data contained in the "src" array. The extent of the array_view is that of the src array, and the origin of the array view is at zero. |                                                                                                          |
| <b>Parameters:</b>                                                                                                                                                                    |                                                                                                          |
| <code>src</code>                                                                                                                                                                      | An array which contains the data that this array_view is bound to.                                       |

|                                                                                                                                                                                                                                                                    |                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>template &lt;typename Container&gt; explicit array_view&lt;T, 1&gt;::array_view(Container&amp; src);</code>                                                                                                                                                  | <code>template &lt;typename Container&gt; explicit array_view&lt;const T, 1&gt;::array_view(const Container&amp; src);</code>                                |
| <code>template &lt;typename value_type, int Size&gt; explicit array_view&lt;T, 1&gt;::array_view(value_type (&amp;src)[Size]) restrict(amp,cpu);</code>                                                                                                            |                                                                                                                                                              |
| <code>template &lt;typename value_type, int Size&gt; explicit array_view&lt;const T, 1&gt;::array_view(const value_type (&amp;src)[Size]) restrict(amp,cpu);</code>                                                                                                |                                                                                                                                                              |
| Constructs a 1D array_view which is bound to the data contained in the "src" container or a 1D C++ array. The extent of the array_view is that given by the "size" of the src container or the size of the C++ array, and the origin of the array view is at zero. |                                                                                                                                                              |
| <b>Parameters:</b>                                                                                                                                                                                                                                                 |                                                                                                                                                              |
| <code>src</code>                                                                                                                                                                                                                                                   | A template argument that must resolve to a linear container that supports .data() and .size() members (such as std::vector or std::array) or a 1D C++ array. |

|                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>template &lt;typename Container&gt; array_view&lt;T,N&gt;::array_view(const extent&lt;N&gt;&amp; extent, Container&amp; src);</code>             |
| <code>template &lt;typename Container&gt; array_view&lt;const T,N&gt;::array_view(const extent&lt;N&gt;&amp; extent, const Container&amp; src);</code> |
| Constructs an array_view which is bound to the data contained in the "src" container. The extent of the array_view is that                             |

given by the "extent" argument, and the origin of the array view is at zero.

**Parameters:**

|               |                                                                                                                                           |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>src</i>    | A template argument that must resolve to a linear container that supports .data() and .size() members (such as std::vector or std::array) |
| <i>extent</i> | The extent of this array_view.                                                                                                            |

2738

```
array_view<T,N>::array_view(const extent<N>& extent, value_type* src) restrict(amp,cpu);
array_view<const T,N>::array_view(const extent<N>& extent, const value_type* src)
restrict(amp,cpu);
```

Constructs an array\_view which is bound to the data contained in the "src" container. The extent of the array\_view is that given by the "extent" argument, and the origin of the array view is at zero.

**Parameters:**

|               |                                                                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>src</i>    | A pointer to the source data this array_view will bind to. If the number of elements pointed to is less than the size of extent, the behavior is undefined. |
| <i>extent</i> | The extent of this array_view.                                                                                                                              |

2739

```
explicit array_view<T,N>::array_view(const extent<N>& extent);
```

Constructs an array\_view which is not bound to a data source. The extent of the array\_view is that given by the "extent" argument, and the origin of the array view is at zero. An array\_view thus constructed represents uninitialized data and the underlying allocations are created lazily as the array\_view is accessed on different locations (on an accelerator\_view or on the CPU).

**Parameters:**

|               |                                |
|---------------|--------------------------------|
| <i>extent</i> | The extent of this array_view. |
|---------------|--------------------------------|

2740

```
template <typename Container>
array_view<T,1>::array_view(int e0, Container& src);
template <typename Container>
array_view<T,2>::array_view(int e0, int e1, Container& src);
template <typename Container>
array_view<T,3>::array_view(int e0, int e1, int e2, Container& src);

template <typename Container>
array_view<const T,1>::array_view(int e0, const Container& src);
template <typename Container>
array_view<const T,2>::array_view(int e0, int e1, const Container& src);
template <typename Container>
array_view<const T,3>::array_view(int e0, int e1, int e2, const Container& src);
```

Equivalent to construction using "array\_view(extent<N>(e0 [, e1 [, e2 ]]), src)".

**Parameters:**

|                          |                                                                                                                                               |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>e0 [, e1 [, e2 ]]</i> | The component values that will form the extent of this array_view.                                                                            |
| <i>src</i>               | A template argument that must resolve to a contiguous container that supports .data() and .size() members (such as std::vector or std::array) |

2741

```
array_view<T,1>::array_view(int e0, value_type* src) restrict(amp,cpu);
array_view<T,2>::array_view(int e0, int e1, value_type* src) restrict(amp,cpu);
array_view<T,3>::array_view(int e0, int e1, int e2, value_type* src) restrict(amp,cpu);

array_view<const T,1>::array_view(int e0, const value_type* src) restrict(amp,cpu);
array_view<const T,2>::array_view(int e0, int e1, const value_type* src) restrict(amp,cpu);
array_view<const T,3>::array_view(int e0, int e1, int e2, const value_type* src)
restrict(amp,cpu);
```

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                |                                                                                                                                                |                              |                                                                                                                                                             |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                | <p>Equivalent to construction using “<code>array_view(extent&lt;N&gt;(e0 [, e1 [, e2 ]]), src)</code>”.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>e0 [, e1 [, e2 ]]</code></td><td>The component values that will form the extent of this array_view.</td></tr> <tr> <td><code>src</code></td><td>A pointer to the source data this array_view will bind to. If the number of elements pointed to is less than the size of extent, the behavior is undefined.</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                   | <code>e0 [, e1 [, e2 ]]</code> | The component values that will form the extent of this array_view.                                                                             | <code>src</code>             | A pointer to the source data this array_view will bind to. If the number of elements pointed to is less than the size of extent, the behavior is undefined. |
| <code>e0 [, e1 [, e2 ]]</code> | The component values that will form the extent of this array_view.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| <code>src</code>               | A pointer to the source data this array_view will bind to. If the number of elements pointed to is less than the size of extent, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| 2742                           | <pre>explicit array_view&lt;T,1&gt;::array_view(int e0); explicit array_view&lt;T,2&gt;::array_view(int e0, int e1); explicit array_view&lt;T,3&gt;::array_view(int e0, int e1, int e2);</pre> <p>Equivalent to construction using “<code>array_view(extent&lt;N&gt;(e0 [, e1 [, e2 ]]))</code>”.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>e0 [, e1 [, e2 ]]</code></td><td>The component values that will form the extent of this array_view.</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                 | <code>e0 [, e1 [, e2 ]]</code> | The component values that will form the extent of this array_view.                                                                             |                              |                                                                                                                                                             |
| <code>e0 [, e1 [, e2 ]]</code> | The component values that will form the extent of this array_view.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| 2743                           | <pre>array_view&lt;T,N&gt;::array_view(const array_view&lt;T,N&gt;&amp; other) restrict(amp,cpu); array_view&lt;const T,N&gt;::array_view(const array_view&lt;const T,N&gt;&amp; other) restrict(amp,cpu);</pre> <p>Copy constructor. Constructs an array_view from the supplied argument other. A shallow copy is performed.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>other</code></td><td>An object of type <code>array_view&lt;T,N&gt;</code> or <code>array_view&lt;const T,N&gt;</code> from which to initialize this new array_view.</td></tr> </table>                                                                                                                                                                                                                                                                                     | <code>other</code>             | An object of type <code>array_view&lt;T,N&gt;</code> or <code>array_view&lt;const T,N&gt;</code> from which to initialize this new array_view. |                              |                                                                                                                                                             |
| <code>other</code>             | An object of type <code>array_view&lt;T,N&gt;</code> or <code>array_view&lt;const T,N&gt;</code> from which to initialize this new array_view.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| 2744                           | <pre>array_view&lt;const T,N&gt;::array_view(const array_view&lt;T,N&gt;&amp; other) restrict(amp,cpu);</pre> <p>Converting constructor. Constructs an array_view from the supplied argument other. A shallow copy is performed.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>other</code></td><td>An object of type <code>array_view&lt;T,N&gt;</code> from which to initialize this new array_view.</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                              | <code>other</code>             | An object of type <code>array_view&lt;T,N&gt;</code> from which to initialize this new array_view.                                             |                              |                                                                                                                                                             |
| <code>other</code>             | An object of type <code>array_view&lt;T,N&gt;</code> from which to initialize this new array_view.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| 2745                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| 2746                           | <b>5.2.3 Members</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| 2747                           | <pre>_declspec(property(get=get_extent)) extent&lt;N&gt; extent; extent&lt;N&gt; get_extent() const restrict(cpu,amp);</pre> <p>Access the extent that defines the shape of this array_view.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| 2748                           | <pre>_declspec(property(get=get_source_accelerator_view)) accelerator_view source_accelerator_view; accelerator_view get_source_accelerator_view() const;</pre> <p>Access the accelerator_view where the data source of the array_view is located.</p> <p>When the data source of the array_view is native CPU memory, the method returns <code>accelerator::cpu_accelerator.default_view</code>. When the data source underlying the array_view is an array, the method returns the <code>accelerator_view</code> where the source array is located.</p>                                                                                                                                                                                                                                                                                                                |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| 2749                           | <pre>array_view&lt;T,N&gt;&amp; array_view&lt;T,N&gt;::operator=(const array_view&lt;T,N&gt;&amp; other) restrict(amp,cpu); array_view&lt;const T,N&gt;&amp; array_view&lt;const T,N&gt;::operator=(const array_view&lt;T,N&gt;&amp; other) restrict(amp,cpu); array_view&lt;const T,N&gt;&amp; array_view&lt;const T,N&gt;::operator=(const array_view&lt;const T,N&gt;&amp; other) restrict(amp,cpu);</pre> <p>Assigns the contents of the array_view “other” to this array_view, using a shallow copy. Both array_views will refer to the same data.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><code>other</code></td><td>An object of type <code>array_view&lt;T,N&gt;</code> from which to copy into this array.</td></tr> </table> <p><b>Return Value:</b></p> <table border="1"> <tr> <td>Returns <code>*this</code>.</td><td></td></tr> </table> | <code>other</code>             | An object of type <code>array_view&lt;T,N&gt;</code> from which to copy into this array.                                                       | Returns <code>*this</code> . |                                                                                                                                                             |
| <code>other</code>             | An object of type <code>array_view&lt;T,N&gt;</code> from which to copy into this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                |                                                                                                                                                |                              |                                                                                                                                                             |
| Returns <code>*this</code> .   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                |                                                                                                                                                |                              |                                                                                                                                                             |

2750

|                                                        |                                                                                                                                |
|--------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <code>void copy_to(array&lt;T,N&gt;&amp; dest);</code> | Copies the data referred to by this array_view to the array given by "dest", as if by calling "copy(*this, dest)" (see 5.3.2). |
|--------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|

**Parameters:**`dest`An object of type `array <T,N>` to which to copy data from this array.

2751

|                                                        |                                                                                                                             |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>void copy_to(const array_view&amp; dest);</code> | Copies the contents of this array_view to the array_view given by "dest", as if by calling "copy(*this, dest)" (see 5.3.2). |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|

**Parameters:**`dest`An object of type `array_view<T,N>` to which to copy data from this array.

2752

|                                                                        |                                                                                    |
|------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| <code>T* array_view&lt;T,1&gt;::data() const restrict(amp,cpu);</code> | <code>const T* array_view&lt;const T,1&gt;::data() const restrict(amp,cpu);</code> |
|------------------------------------------------------------------------|------------------------------------------------------------------------------------|

Returns a pointer to the first data element underlying this array\_view. This is only available on array\_views of rank 1.

When the data source of the array\_view is native CPU memory, the pointer returned by data() is valid for the lifetime of the data source.

When the data source underlying the array\_view is an array, or the array view is created without a data source, the pointer returned by data() in CPU context is ephemeral and is invalidated when the original data source or any of its views are accessed on an accelerator\_view through a parallel\_for\_each or a copy operation.

**Return Value:**

A (const) pointer to the first element in the linearized array.

2753

|                                    |  |
|------------------------------------|--|
| <code>void refresh() const;</code> |  |
|------------------------------------|--|

Calling this member function informs the array\_view that its bound memory has been modified outside the array\_view interface. This will render all cached information stale.

2754

|                                                                                                   |                                                                      |
|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| <code>void array_view&lt;T, N&gt;::synchronize(access_type type = access_type_read) const;</code> | <code>void array_view&lt;const T, N&gt;::synchronize() const;</code> |
|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|

Calling this member function synchronizes any modifications made to the data underlying "this" array\_view to its source data container. For example, for an array\_view on system memory, if the data underlying the view are modified on a remote accelerator\_view through a parallel\_for\_each invocation, calling synchronize ensures that the modifications are synchronized to the source data and will be visible through the system memory pointer which the array\_view was created over.

For writable array\_view objects, callers of this functional can optionally specify the type of access desired on the source data container through the "type" parameter. For example specifying a "access\_type\_read" (which is also the default value of the parameter) indicates that the data has been synchronized to its source location only for reading. On the other hand, specifying an access\_type of "access\_type\_read\_write" synchronizes the data to its source location both for reading and writing; i.e. any modifications to the source data directly through the source data container are legal after synchronizing the array\_view with write access and before subsequently accessing the array\_view on another remote location.

It is advisable to be precise about the access\_type specified in the synchronize call; i.e. if only write access is required, specifying access\_type\_write may yield better performance than calling synchronize with "access\_type\_read\_write" since the latter may require any modifications made to the data on remote locations to be synchronized to the source location, which is unnecessary if the contents are intended to be overwritten without reading.

**Parameters:**`type`

An argument of type "access\_type" which specifies the type of access on the data source that the array\_view is synchronized for.

2755

|                                                                                                                      |                                                                                         |
|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <code>completion_future array_view&lt;T, N&gt;::synchronize_async(access_type type = access_type_read) const;</code> | <code>completion_future array_view&lt;const T, N&gt;::synchronize_async() const;</code> |
|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|

An asynchronous version of `synchronize`, which returns a completion future object. When the future is ready, the synchronization operation is complete.**Return Value:**An object of type `completion_future` that can be used to determine the status of the asynchronous operation or can be used to chain other operations to be executed after the completion of the asynchronous operation.

2756

```
void array_view<T, N>::synchronize_to(const accelerator_view& av, access_type type =
access_type_read) const;
void array_view<const T, N>::synchronize_to(const accelerator_view& av) const;
```

Calling this member function synchronizes any modifications made to the data underlying "this" array\_view to the specified accelerator\_view "av". For example, for an array\_view on system memory, if the data underlying the view is modified on the CPU, and synchronize\_to is called on "this" array\_view, then the array\_view contents are cached on the specified accelerator\_view location.

For writable array\_view objects, callers of this functional can optionally specify the type of access desired on the specified target accelerator\_view "av", through the "type" parameter. For example specifying a "access\_type\_read" (which is also the default value of the parameter) indicates that the data has been synchronized to "av" only for reading. On the other hand, specifying an access\_type of "access\_type\_read\_write" synchronizes the data to "av" both for reading and writing; i.e. any modifications to the data on "av" are legal after synchronizing the array\_view with write access and before subsequently accessing the array\_view on a location other than "av".

It is advisable to be precise about the access\_type specified in the synchronize call; i.e. if only write access is required, specifying access\_type\_write may yield better performance than calling synchronize with "access\_type\_read\_write" since the later may require any modifications made to the data on remote locations to be synchronized to "av", which is unnecessary if the contents are intended to be immediately overwritten without reading.

#### Parameters:

|      |                                                                                                                                  |
|------|----------------------------------------------------------------------------------------------------------------------------------|
| av   | The target accelerator_view that "this" array_view is synchronized for access on.                                                |
| type | An argument of type "access_type" which specifies the type of access on the data source that the array_view is synchronized for. |

2757

```
completion_future array_view<T, N>::synchronize_to_async(const accelerator_view& av, access_type
type = access_type_read) const;
completion_future array_view<const T, N>::synchronize_to_async(const accelerator_view& av)
const;
```

An asynchronous version of [synchronize\\_to](#), which returns a completion future object. When the future is ready, the synchronization operation is complete.

#### Parameters:

|      |                                                                                                                                  |
|------|----------------------------------------------------------------------------------------------------------------------------------|
| av   | The target accelerator_view that "this" array_view is synchronized for access on.                                                |
| type | An argument of type "access_type" which specifies the type of access on the data source that the array_view is synchronized for. |

#### Return Value:

An object of type completion\_future that can be used to determine the status of the asynchronous operation or can be used to chain other operations to be executed after the completion of the asynchronous operation.

2758

```
void array_view<T, N>::discard_data() const;
```

Indicates to the runtime that it may discard the current logical contents of this array\_view. This is an optimization hint to the runtime used to avoid copying the current contents of the view to a target accelerator\_view, and its use is recommended if the existing content is not needed.

2759

## 5.2.4 Indexing

2760

Accessing an [array\\_view](#) out of bounds yields undefined results.

2761

```
T& array_view<T,N>::operator[](const index<N>& idx) const restrict(amp,cpu);
T& array_view<T,N>::operator()(const index<N>& idx) const restrict(amp,cpu);
```

Returns a reference to the element of this array\_view that is at the location in N-dimensional space specified by "idx".

#### Parameters:

|     |                                                                                           |
|-----|-------------------------------------------------------------------------------------------|
| idx | An object of type <code>index&lt;N&gt;</code> that specifies the location of the element. |
|-----|-------------------------------------------------------------------------------------------|

2762

```
const T& array_view<const T,N>::operator[](const index<N>& idx) const restrict(amp,cpu);
const T& array_view<const T,N>::operator()(const index<N>& idx) const restrict(amp,cpu);
```

|                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                          |                                                                                           |                                                                                  |                                                                                                |
|----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
|                                                                                  | Returns a const reference to the element of this array_view that is at the location in N-dimensional space specified by "idx".                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                          |                                                                                           |                                                                                  |                                                                                                |
| 2765                                                                             | <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>idx</i></td><td>An object of type <code>index&lt;N&gt;</code> that specifies the location of the element.</td></tr> </table> <pre>T&amp; array_view&lt;T,N&gt;::get_ref(const index&lt;N&gt;&amp; idx) const restrict(amp,cpu); const T&amp; array_view&lt;const T,N&gt;::get_ref(const index&lt;N&gt;&amp; idx) const restrict(amp,cpu);</pre> <p>Returns a reference to the element of this array_view that is at the location in N-dimensional space specified by "idx".</p> <p>Unlike the other indexing operators for accessing the array_view on the CPU, this method does not implicitly synchronize this array_view's contents to the CPU. After accessing the array_view on a remote location or performing a copy operation involving this array_view, users are responsible to explicitly synchronize the array_view to the CPU before calling this method. Failure to do so results in undefined behavior.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>idx</i></td><td>An object of type <code>index&lt;N&gt;</code> from that specifies the location of the element.</td></tr> </table> | <i>idx</i>               | An object of type <code>index&lt;N&gt;</code> that specifies the location of the element. | <i>idx</i>                                                                       | An object of type <code>index&lt;N&gt;</code> from that specifies the location of the element. |
| <i>idx</i>                                                                       | An object of type <code>index&lt;N&gt;</code> that specifies the location of the element.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                          |                                                                                           |                                                                                  |                                                                                                |
| <i>idx</i>                                                                       | An object of type <code>index&lt;N&gt;</code> from that specifies the location of the element.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                          |                                                                                           |                                                                                  |                                                                                                |
| 2766                                                                             | <pre>T&amp; array_view&lt;T,1&gt;::operator[](int i0) const restrict(amp,cpu); T&amp; array_view&lt;T,1&gt;::operator()(int i0) const restrict(amp,cpu); T&amp; array_view&lt;T,2&gt;::operator()(int i0, int i1) const restrict(amp,cpu); T&amp; array_view&lt;T,3&gt;::operator()(int i0, int i1, int i2) const restrict(amp,cpu);</pre> <p>Equivalent to <code>"array_view&lt;T,N&gt;::operator()(index&lt;N&gt;(i0 [, i1 [, i2 ]]))"</code>.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>i0 [, i1 [, i2 ]]</i></td><td>The component values that will form the index into this array.</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | <i>i0 [, i1 [, i2 ]]</i> | The component values that will form the index into this array.                            |                                                                                  |                                                                                                |
| <i>i0 [, i1 [, i2 ]]</i>                                                         | The component values that will form the index into this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                          |                                                                                           |                                                                                  |                                                                                                |
| 2767                                                                             | <pre>const T&amp; array_view&lt;const T,1&gt;::operator[](int i0) const restrict(amp,cpu); const T&amp; array_view&lt;const T,1&gt;::operator()(int i0) const restrict(amp,cpu); const T&amp; array_view&lt;const T,2&gt;::operator()(int i0, int i1) const restrict(amp,cpu); const T&amp; array_view&lt;const T,3&gt;::operator()(int i0, int i1, int i2) const restrict(amp,cpu);</pre> <p>Equivalent to <code>"array_view&lt;const T,N&gt;::operator()(index&lt;N&gt;(i0 [, i1 [, i2 ]])) const"</code>.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>i0 [, i1 [, i2 ]]</i></td><td>The component values that will form the index into this array.</td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | <i>i0 [, i1 [, i2 ]]</i> | The component values that will form the index into this array.                            |                                                                                  |                                                                                                |
| <i>i0 [, i1 [, i2 ]]</i>                                                         | The component values that will form the index into this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                          |                                                                                           |                                                                                  |                                                                                                |
| 2768                                                                             | <pre>array_view&lt;T,N-1&gt; array_view&lt;T,N&gt;::operator[](int i0) const restrict(amp,cpu); array_view&lt;const T,N-1&gt; array_view&lt;const T,N&gt;::operator[](int i0) const restrict(amp,cpu); array_view&lt;T,N-1&gt; array_view&lt;T,N&gt;::operator()(int i0) const restrict(amp,cpu); array_view&lt;const T,N-1&gt; array_view&lt;const T,N&gt;::operator()(int i0) const restrict(amp,cpu);</pre> <p>This overload is defined for <code>array_view&lt;T,N&gt;</code> where <math>N \geq 2</math>.</p> <p>This mode of indexing is equivalent to projecting on the most-significant dimension. It allows C-style indexing. For example:</p> <pre>array&lt;float,4&gt; myArray(myExtents, ...);  myArray[index&lt;4&gt;(5,4,3,2)] = 7; assert(myArray[5][4][3][2] == 7);</pre> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>i0</i></td><td>An integer that is the index into the most-significant dimension of this array.</td></tr> </table> <p><b>Return Value:</b></p> <table border="1"> <tr> <td>Returns an array_view whose dimension is one lower than that of this array_view.</td></tr> </table>                                               | <i>i0</i>                | An integer that is the index into the most-significant dimension of this array.           | Returns an array_view whose dimension is one lower than that of this array_view. |                                                                                                |
| <i>i0</i>                                                                        | An integer that is the index into the most-significant dimension of this array.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                          |                                                                                           |                                                                                  |                                                                                                |
| Returns an array_view whose dimension is one lower than that of this array_view. |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                          |                                                                                           |                                                                                  |                                                                                                |
| 2769                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                          |                                                                                           |                                                                                  |                                                                                                |
| 2770                                                                             | <h2>5.2.5 View Operations</h2>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                          |                                                                                           |                                                                                  |                                                                                                |
| 2771                                                                             | <pre>array_view section(const index&lt;N&gt;&amp; origin, const extent&lt;N&gt;&amp; ext) const restrict(amp,cpu);</pre> <p>Returns a subsection of the source array view at the origin specified by "idx" and with the extent specified by "ext"</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                          |                                                                                           |                                                                                  |                                                                                                |

Example:

```
array<float,2> a(extent<2>(200,100));
array_view<float,2> v1(a); // v1.extent = <200,100>
array_view<float,2> v2 = v1.section(index<2>(15,25), extent<2>(40,50));
assert(v2(0,0) == v1(15,25));
```

**Parameters:**

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>origin</i> | Provides the offset/origin of the resulting section. |
|---------------|------------------------------------------------------|

|            |                                               |
|------------|-----------------------------------------------|
| <i>ext</i> | Provides the extent of the resulting section. |
|------------|-----------------------------------------------|

**Return Value:**

Returns a subsection of the source array at specified origin, and with the specified extent.

2772

|                                                                     |
|---------------------------------------------------------------------|
| array_view section(const index<N>& origin) const restrict(amp,cpu); |
|---------------------------------------------------------------------|

|                                                   |
|---------------------------------------------------|
| Equivalent to "section(idx, this->extent - idx)". |
|---------------------------------------------------|

2773

2774

|                                                                   |
|-------------------------------------------------------------------|
| array_view section(const extent<N>& ext) const restrict(amp,cpu); |
|-------------------------------------------------------------------|

|                                           |
|-------------------------------------------|
| Equivalent to "section(index<N>(), ext)". |
|-------------------------------------------|

2775

2776

|                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| array_view<T,1> array_view<T,1>::section(int i0, int e0) const restrict(amp,cpu);<br>array_view<const T,1> array_view<const T,1>::section(int i0, int e0) const restrict(amp,cpu); |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                                                      |
|------------------------------------------------------------------------------------------------------|
| array_view<T,2> array_view<T,2>::section(int i0, int i1, int e0, int e1) const<br>restrict(amp,cpu); |
|------------------------------------------------------------------------------------------------------|

|                                                                                                                  |
|------------------------------------------------------------------------------------------------------------------|
| array_view<const T,2> array_view<const T,2>::section(int i0, int i1,<br>int e0, int e1) const restrict(amp,cpu); |
|------------------------------------------------------------------------------------------------------------------|

|                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------|
| array_view<T,3> array_view<T,3>::section(int i0, int i1, int i2,<br>int e0, int e1, int e2) const restrict(amp,cpu); |
|----------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------|
| array_view<const T,3> array_view<const T,3>::section(int i0, int i1, int i2,<br>int e0, int e1, int e2) const restrict(amp,cpu); |
|----------------------------------------------------------------------------------------------------------------------------------|

|                                                                                      |
|--------------------------------------------------------------------------------------|
| Equivalent to "section(index<N>(i0 [, i1 [, i2 ]]), extent<N>(e0 [, e1 [, e2 []]))". |
|--------------------------------------------------------------------------------------|

**Parameters:**

|                          |                                                               |
|--------------------------|---------------------------------------------------------------|
| <i>i0 [, i1 [, i2 ]]</i> | The component values that will form the origin of the section |
|--------------------------|---------------------------------------------------------------|

|                          |                                                               |
|--------------------------|---------------------------------------------------------------|
| <i>e0 [, e1 [, e2 ]]</i> | The component values that will form the extent of the section |
|--------------------------|---------------------------------------------------------------|

2777

|                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------|
| template<typename ElementType><br>array_view<ElementType,1> array_view<T,1>::reinterpret_as() const restrict(amp,cpu); |
|------------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------|
| template<typename ElementType><br>array_view<const ElementType,1> array_view<const T,1>::reinterpret_as() const<br>restrict(amp,cpu); |
|---------------------------------------------------------------------------------------------------------------------------------------|

This member function is similar to "array<T,N>::reinterpret\_as" (see 5.1.5), although it only supports array\_views of rank 1 (only those guarantee that all elements are laid out contiguously).

The size of the reinterpreted ElementType must evenly divide into the total size of this array\_view.

**Return Value:**

|                                                                                                            |
|------------------------------------------------------------------------------------------------------------|
| Returns an array_view from this array_view<T,1> with the element type reinterpreted from T to ElementType. |
|------------------------------------------------------------------------------------------------------------|

2778

|                                                                                                                    |
|--------------------------------------------------------------------------------------------------------------------|
| template <int K><br>array_view<T,K> array_view<T,1>::view_as(const extent<K>& viewExtent) const restrict(amp,cpu); |
|--------------------------------------------------------------------------------------------------------------------|

```
template <int K>
    array_view<const T,K> array_view<const T,1>::view_as(const extent<K>& viewExtent) const
restrict(amp,cpu);
```

This member function is similar to `array<T,N>::view_as`" (see 5.1.5), although it only supports array\_views of rank 1 (only those guarantee that all elements are laid out contiguously).

**Return Value:**

Returns an `array_view` from this `array_view<T,1>` with the rank changed to K from 1.

2779

## 5.3 Copying Data

2781

2782 C++ AMP offers a set of `copy` functions which covers all synchronous data transfer requirements. In all cases, copying data  
 2783 is not supported while executing on an accelerator (in other words, the copy functions do not have a `restrict(amp)` clause).  
 2784 The general form of copy is:

2785

```
copy(src, dest);
```

2787

**Informative:** Note that this more closely follows the STL convention (destination is the last argument, as in `std::copy`) and is  
 opposite of the C-style convention (destination is the first argument, as in `memcpy`).

2790

2791

Copying to `array` and `array_view` types is supported from the following sources:

2792

- An `array` or `array_view` with the same rank and element type (apart from `const` qualifier) as the destination `array` or `array_view`.
- An InputIterator or a pair of thereof whose value type is convertible to the element type of the destination `array` or `array_view`.

2796

**Informative:** Iterators referring to a contiguous memory (e.g. obtained from `std::vector`) can be handled more efficiently.

2797

2798 Copying from `array` and `array_view` types is supported to the destination being an OutputIterator whose value type is  
 2799 convertible from the element type of the source.

2800

2801 The copy operation always performs a deep copy.

2802

2803 Asynchronous copy has the same semantics as synchronous copy, except that they return a `completion_future` that can  
 2804 be waited on.

2805

2806

### 5.3.1 Synopsis

2807

```
template <typename T, int N>
void copy(const array<T,N>& src, array<T,N>& dest);
template <typename T, int N>
void copy(const array<T,N>& src, const array_view<T,N>& dest);

template <typename T, int N>
void copy(const array_view<const T,N>& src, array<T,N>& dest);
template <typename T, int N>
void copy(const array_view<const T,N>& src, const array_view<T,N>& dest);

template <typename T, int N>
void copy(const array_view<T,N>& src, array<T,N>& dest);
template <typename T, int N>
void copy(const array_view<T,N>& src, const array_view<T,N>& dest);
```

```

2823 template <typename InputIter, typename T, int N>
2824     void copy(InputIter srcBegin, InputIter srcEnd, array<T,N>& dest);
2825 template <typename InputIter, typename T, int N>
2826     void copy(InputIter srcBegin, InputIter srcEnd, const array_view<T,N>& dest);
2827
2828 template <typename InputIter, typename T, int N>
2829     void copy(InputIter srcBegin, array<T,N>& dest);
2830 template <typename InputIter, typename T, int N>
2831     void copy(InputIter srcBegin, const array_view<T,N>& dest);
2832
2833 template <typename OutputIter, typename T, int N>
2834     void copy(const array<T,N>& src, OutputIter destBegin);
2835 template <typename OutputIter, typename T, int N>
2836     void copy(const array_view<T,N>& src, OutputIter destBegin);
2837
2838 template <typename T, int N>
2839     completion_future copy_async(const array<T,N>& src, array<T,N>& dest);
2840 template <typename T, int N>
2841     completion_future copy_async(const array<T,N>& src, const array_view<T,N>& dest);
2842
2843 template <typename T, int N>
2844     completion_future copy_async(const array_view<const T,N>& src, array<T,N>& dest);
2845 template <typename T, int N>
2846     completion_future copy_async(const array_view<const T,N>& src, const array_view<T,N>& dest);
2847
2848 template <typename T, int N>
2849     completion_future copy_async(const array_view<T,N>& src, array<T,N>& dest);
2850 template <typename T, int N>
2851     completion_future copy_async(const array_view<T,N>& src, const array_view<T,N>& dest);
2852
2853 template <typename InputIter, typename T, int N>
2854     completion_future copy_async(InputIter srcBegin, InputIter srcEnd, array<T,N>& dest);
2855 template <typename InputIter, typename T, int N>
2856     completion_future copy_async(InputIter srcBegin, InputIter srcEnd, const array_view<T,N>& dest);
2857
2858 template <typename InputIter, typename T, int N>
2859     completion_future copy_async(InputIter srcBegin, array<T,N>& dest);
2860 template <typename InputIter, typename T, int N>
2861     completion_future copy_async(InputIter srcBegin, const array_view<T,N>& dest);
2862
2863 template <typename OutputIter, typename T, int N>
2864     completion_future copy_async(const array<T,N>& src, OutputIter destBegin);
2865 template <typename OutputIter, typename T, int N>
2866     completion_future copy_async(const array_view<T,N>& src, OutputIter destBegin);
2867
2868

```

### 2869 5.3.2 Copying between array and array\_view

2870

2871 An *array<T,N>* can be copied to an object of type *array\_view<T,N>*, and vice versa.

2872

```

template <typename T, int N>
void copy(const array<T,N>& src, array<T,N>& dest);

template <typename T, int N>
completion_future copy_async(const array<T,N>& src, array<T,N>& dest);

```

The contents of "src" are copied into "dest". The source and destination may reside on different accelerators. If the extents

|                                                                 |                                                                    |
|-----------------------------------------------------------------|--------------------------------------------------------------------|
| of "src" and "dest" don't match, a runtime exception is thrown. |                                                                    |
| <b>Parameters:</b>                                              |                                                                    |
| <i>src</i>                                                      | An object of type <code>array&lt;T,N&gt;</code> to be copied from. |
| <i>dest</i>                                                     | An object of type <code>array&lt;T,N&gt;</code> to be copied to.   |

2873

```
template <typename T, int N>
void copy(const array<T,N>& src, const array_view<T,N>& dest);

template <typename T, int N>
completion_future copy_async(const array<T,N>& src, const array_view<T,N>& dest);
```

The contents of "src" are copied into "dest". If the extents of "src" and "dest" don't match, a runtime exception is thrown.

|                    |                                                                       |
|--------------------|-----------------------------------------------------------------------|
| <b>Parameters:</b> |                                                                       |
| <i>src</i>         | An object of type <code>array&lt;T,N&gt;</code> to be copied from.    |
| <i>dest</i>        | An object of type <code>array_view&lt;T,N&gt;</code> to be copied to. |

2874

```
template <typename T, int N>
void copy(const array_view<const T,N>& src, array<T,N>& dest);

template <typename T, int N>
void copy(const array_view<T,N>& src, array<T,N>& dest);

template <typename T, int N>
completion_future copy_async(const array_view<const T,N>& src, array<T,N>& dest);

template <typename T, int N>
completion_future copy_async(const array_view<T,N>& src, array<T,N>& dest);
```

The contents of "src" are copied into "dest". If the extents of "src" and "dest" don't match, a runtime exception is thrown.

|                    |                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters:</b> |                                                                                                                        |
| <i>src</i>         | An object of type <code>array_view&lt;T,N&gt;</code> (or <code>array_view&lt;const T,N&gt;</code> ) to be copied from. |
| <i>dest</i>        | An object of type <code>array&lt;T,N&gt;</code> to be copied to.                                                       |

2875

```
template <typename T, int N>
void copy(const array_view<const T,N>& src, const array_view<T,N>& dest);

template <typename T, int N>
void copy(const array_view<T,N>& src, const array_view<T,N>& dest);

template <typename T, int N>
completion_future copy_async(const array_view<const T,N>& src, const array_view<T,N>& dest);

template <typename T, int N>
completion_future copy_async(const array_view<T,N>& src, const array_view<T,N>& dest);
```

The contents of "src" are copied into "dest". If the extents of "src" and "dest" don't match, a runtime exception is thrown.

|                    |                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters:</b> |                                                                                                                        |
| <i>src</i>         | An object of type <code>array_view&lt;T,N&gt;</code> (or <code>array_view&lt;const T,N&gt;</code> ) to be copied from. |
| <i>dest</i>        | An object of type <code>array_view&lt;T,N&gt;</code> to be copied to.                                                  |

2876  
28772878 **5.3.3 Copying from standard containers to arrays or array\_views**

2879

2880 A standard container can be copied into an [array](#) or [array\\_view](#) by specifying an iterator range.2881 ***Informative:*** Standard containers that guarantee a contiguous memory allocation (such as `std::vector` and `std::array`) can be  
2882 handled very efficiently.

2883

```
template <typename InputIter, typename T, int N>
void copy(InputIter srcBegin, InputIter srcEnd, array<T,N>& dest);

template <typename InputIter, typename T, int N>
void copy(InputIter srcBegin, array<T,N>& dest);

template <typename InputIter, typename T, int N>
completion_future copy_async(InputIter srcBegin, InputIter srcEnd, array<T,N>& dest);

template <typename InputIter, typename T, int N>
completion_future copy_async(InputIter srcBegin, array<T,N>& dest);
```

The contents of a source container from the iterator range `[srcBegin,srcEnd)` are copied into “dest”. If the number of elements in the iterator range is not equal to “`dest.extent.size()`”, an exception is thrown.

In the overloads which don’t take an end-iterator it is assumed that the source iterator is able to provide at least `dest.extent.size()` elements, but no checking is performed (nor possible).

**Parameters:**

|                       |                                                         |
|-----------------------|---------------------------------------------------------|
| <code>srcBegin</code> | An iterator to the first element of a source container. |
|-----------------------|---------------------------------------------------------|

|                     |                                               |
|---------------------|-----------------------------------------------|
| <code>srcEnd</code> | An iterator to the end of a source container. |
|---------------------|-----------------------------------------------|

|                   |                                                                  |
|-------------------|------------------------------------------------------------------|
| <code>dest</code> | An object of type <code>array&lt;T,N&gt;</code> to be copied to. |
|-------------------|------------------------------------------------------------------|

2884

```
template <typename InputIter, typename T, int N>
void copy(InputIter srcBegin, InputIter srcEnd, const array_view<T,N>& dest);

template <typename InputIter, typename T, int N>
void copy(InputIter srcBegin, const array_view<T,N>& dest);

template <typename InputIter, typename T, int N>
completion_future copy_async(InputIter srcBegin, InputIter srcEnd, const array_view<T,N>& dest);

template <typename InputIter, typename T, int N>
completion_future copy_async(InputIter srcBegin, const array_view<T,N>& dest);
```

The contents of a source container from the iterator range `[srcBegin,srcEnd)` are copied into “dest”. If the number of elements in the iterator range is not equal to “`dest.extent.size()`”, an exception is thrown.

In the overloads which don’t take an end-iterator it is assumed that the source iterator is able to provide at least `dest.extent.size()` elements, but no checking is performed (nor possible).

**Parameters:**

|                       |                                                         |
|-----------------------|---------------------------------------------------------|
| <code>srcBegin</code> | An iterator to the first element of a source container. |
|-----------------------|---------------------------------------------------------|

|                     |                                               |
|---------------------|-----------------------------------------------|
| <code>srcEnd</code> | An iterator to the end of a source container. |
|---------------------|-----------------------------------------------|

|             |                                                                       |
|-------------|-----------------------------------------------------------------------|
| <i>dest</i> | An object of type <code>array_view&lt;T,N&gt;</code> to be copied to. |
|-------------|-----------------------------------------------------------------------|

2885

### 2886 5.3.4 Copying from arrays or array\_views to standard containers

2887

2888 An array or array\_view can be copied into a standard container by specifying the begin iterator.

2889 *Informative:* Standard containers that guarantee a contiguous memory allocation (such as `std::vector` and `std::array`) can be  
2890 handled very efficiently.

2891

```
template <typename OutputIter, typename T, int N>
void copy(const array<T,N>& src, OutputIter destBegin);

template <typename OutputIter, typename T, int N>
completion_future copy_async(const array<T,N>& src, OutputIter destBegin);
```

The contents of a source array are copied into "dest" starting with iterator destBegin. If the number of elements in the range starting destBegin in the destination container is smaller than "src.extent.size()", the behavior is undefined.

**Parameters:**

|            |                                                                    |
|------------|--------------------------------------------------------------------|
| <i>src</i> | An object of type <code>array&lt;T,N&gt;</code> to be copied from. |
|------------|--------------------------------------------------------------------|

|                  |                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------|
| <i>destBegin</i> | An output iterator addressing the position of the first element in the destination container. |
|------------------|-----------------------------------------------------------------------------------------------|

2892

```
template <typename OutputIter, typename T, int N>
void copy(const array_view<T,N>& src, OutputIter destBegin);

template <typename OutputIter, typename T, int N>
completion_future copy_async(const array_view<T,N>& src, OutputIter destBegin);
```

The contents of a source array are copied into "dest" starting with iterator destBegin. If the number of elements in the range starting destBegin in the destination container is smaller than "src.extent.size()", the behavior is undefined.

**Parameters:**

|            |                                                                         |
|------------|-------------------------------------------------------------------------|
| <i>src</i> | An object of type <code>array_view&lt;T,N&gt;</code> to be copied from. |
|------------|-------------------------------------------------------------------------|

|                  |                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------|
| <i>destBegin</i> | An output iterator addressing the position of the first element in the destination container. |
|------------------|-----------------------------------------------------------------------------------------------|

2893

## 2894 6 Atomic Operations

2895 C++ AMP provides a set of atomic operations in the `concurrency` namespace. These operations are applicable in  
2896 `restrict(amp)` contexts and may be applied to memory locations within `concurrency::array` instances and to memory  
2897 locations within `tile_static` variables. Section 3 provides a full description of the C++ AMP memory model and how atomic  
2898 operations fit into it.

### 2899 6.1 Synopsis

2900

```
2901 int atomic_exchange(int * dest, int val) restrict(amp);
2902 unsigned int atomic_exchange(unsigned int * dest, unsigned int val) restrict(amp);
2903 float atomic_exchange(float * dest, float val) restrict(amp);
2904
2905 bool atomic_compare_exchange(int * dest, int * expected_value, int val) restrict(amp);
2906 bool atomic_compare_exchange(unsigned int * dest, unsigned int * expected_value, unsigned int
2907 val) restrict(amp);
```

```

2909 int atomic_fetch_add(int * dest, int val) restrict(amp);
2910 unsigned int atomic_fetch_add(unsigned int * dest, unsigned int val) restrict(amp);
2911
2912 int atomic_fetch_sub(int * dest, int val) restrict(amp);
2913 unsigned int atomic_fetch_sub(unsigned int * dest, unsigned int val) restrict(amp);
2914
2915 int atomic_fetch_max(int * dest, int val) restrict(amp);
2916 unsigned int atomic_fetch_max(unsigned int * dest, unsigned int val) restrict(amp);
2917
2918 int atomic_fetch_min(int * dest, int val) restrict(amp);
2919 unsigned int atomic_fetch_min(unsigned int * dest, unsigned int val) restrict(amp);
2920
2921 int atomic_fetch_and(int * dest, int val) restrict(amp);
2922 unsigned int atomic_fetch_and(unsigned int * dest, unsigned int val) restrict(amp);
2923
2924 int atomic_fetch_or(int * dest, int val) restrict(amp);
2925 unsigned int atomic_fetch_or(unsigned int * dest, unsigned int val) restrict(amp);
2926
2927 int atomic_fetch_xor(int * dest, int val) restrict(amp);
2928 unsigned int atomic_fetch_xor(unsigned int * dest, unsigned int val) restrict(amp);
2929
2930 int atomic_fetch_inc(int * dest) restrict(amp);
2931 unsigned int atomic_fetch_inc(unsigned int * dest) restrict(amp);
2932
2933 int atomic_fetch_dec(int * dest) restrict(amp);
2934 unsigned int atomic_fetch_dec(unsigned int * dest) restrict(amp);
2935

```

## 6.2 Atomically Exchanging Values

2936

```

int atomic_exchange(int * dest, int val) restrict(amp);
unsigned int atomic_exchange(unsigned int * dest, unsigned int val) restrict(amp);
float atomic_exchange(float * dest, float val) restrict(amp);

```

Atomically read the value stored in *dest*, replace it with the value given in *val* and return the old value to the caller. This function provides overloads for *int*, *unsigned int* and *float* parameters.

**Parameters:**

|             |                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dest</i> | A pointer to the location which needs to be atomically modified. The location may reside within a <i>concurrency::array</i> or <i>concurrency::array_view</i> or within a <i>tile_static</i> variable. |
| <i>val</i>  | The new value to be stored in the location pointed to be <i>dest</i> .                                                                                                                                 |

**Return value:**

These functions return the old value which was previously stored at *dest*, and that was atomically replaced. These functions always succeed.

2938

```

bool atomic_compare_exchange(int * dest, int * expected_val, int val) restrict(amp);
bool atomic_compare_exchange(unsigned int * dest, unsigned int * expected_val, unsigned int val)
restrict(amp);

```

These functions attempt to perform these three steps atomically:

1. Read the value stored in the location pointed to by *dest*
2. Compare the value read in the previous step with the value contained in the location pointed by *expected\_val*
3. Carry the following operations depending on the result of the comparison of the previous step:
  - a. If the values are identical, then the function tries to atomically change the value pointed by *dest* to the value in *val*. The function indicates by its return value whether this transformation has been successful or not.

- b. If the values are not identical, then the function stores the value read in step (1) into the location pointed to by *expected\_val*, and returns *false*.

In terms of sequential semantics, these functions are equivalent to the following pseudo-code:

```
auto t = *dest;
bool eq = t == *expected_val;
if (eq)
    *dest = val;
*expected_val = t;
return eq;
```

These functions may fail spuriously. It is guaranteed that the system as a whole will make progress when threads are contending to atomically modify a variable, but there is no upper bound on the number of failed attempts that any particular thread may experience.

**Parameters:**

|                     |                                                                                                                                                                                                                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>dest</i>         | An pointer to the location which needs to be atomically modified. The location may reside within a <i>concurrency::array</i> or <i>concurrency::array_view</i> or within a <i>tile_static</i> variable.                                                                                                 |
| <i>expected_val</i> | A pointer to a local variable or function parameter. Upon calling the function, the location pointed by <i>expected_val</i> contains the value the caller expects <i>dest</i> to contain. Upon return from the function, <i>expected_val</i> will contain the most recent value read from <i>dest</i> . |
| <i>val</i>          | The new value to be stored in the location pointed to be <i>dest</i> .                                                                                                                                                                                                                                  |

**Return value:**

The return value indicates whether the function has been successful in atomically reading, comparing and modifying the contents of the memory location.

### 2939 6.3 Atomically Applying an Integer Numerical Operation

2940

```
int atomic_fetch_add(int * dest, int val) restrict(amp);
unsigned int atomic_fetch_add(unsigned int * dest, unsigned int val) restrict(amp);

int atomic_fetch_sub(int * dest, int val) restrict(amp);
unsigned int atomic_fetch_sub(unsigned int * dest, unsigned int val) restrict(amp);

int atomic_fetch_max(int * dest, int val) restrict(amp);
unsigned int atomic_fetch_max(unsigned int * dest, unsigned int val) restrict(amp);

int atomic_fetch_min(int * dest, int val) restrict(amp);
unsigned int atomic_fetch_min(unsigned int * dest, unsigned int val) restrict(amp);

int atomic_fetch_and(int * dest, int val) restrict(amp);
unsigned int atomic_fetch_and(unsigned int * dest, unsigned int val) restrict(amp);

int atomic_fetch_or(int * dest, int val) restrict(amp);
unsigned int atomic_fetch_or(unsigned int * dest, unsigned int val) restrict(amp);

int atomic_fetch_xor(int * dest, int val) restrict(amp);
unsigned int atomic_fetch_xor(unsigned int * dest, unsigned int val) restrict(amp);
```

Atomically read the value stored in *dest*, apply the binary numerical operation specific to the function with the read value and *val* serving as input operands, and store the result back to the location pointed by *dest*.

In terms of sequential semantics, the operation performed by any of the above function is described by the following piece of pseudo-code:

```
*dest = *dest ⊗ val;
```

Where the operation denoted by  $\otimes$  is one of: addition (`atomic_fetch_add`), subtraction (`atomic_fetch_sub`), find maximum (`atomic_fetch_max`), find minimum (`atomic_fetch_min`), bit-wise AND (`atomic_fetch_and`), bit-wise OR (`atomic_fetch_or`), bit-wise XOR (`atomic_fetch_xor`).

**Parameters:**

|                   |                                                                                                                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>dest</code> | An pointer to the location which needs to be atomically modified. The location may reside within a <code>concurrency::array</code> or <code>concurrency::array_view</code> or within a <code>tile_static</code> variable. |
| <code>val</code>  | The second operand which participates in the calculation of the binary operation whose result is stored into the location pointed to be <code>dest</code> .                                                               |

**Return value:**

These functions return the old value which was previously stored at `dest`, and that was atomically replaced. These functions always succeed.

2941

```
int atomic_fetch_inc(int * dest) restrict(amp);
unsigned int atomic_fetch_inc(unsigned int * dest) restrict(amp);

int atomic_fetch_dec(int * dest) restrict(amp);
unsigned int atomic_fetch_dec(unsigned int * dest) restrict(amp);
```

Atomically increment or decrement the value stored at the location point to by `dest`.

**Parameters:**

|                   |                                                                                                                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>dest</code> | An pointer to the location which needs to be atomically modified. The location may reside within a <code>concurrency::array</code> or <code>concurrency::array_view</code> or within a <code>tile_static</code> variable. |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Return value:**

These functions return the old value which was previously stored at `dest`, and that was atomically replaced. These functions always succeed.

2942

## 7 Launching Computations: parallel\_for\_each

2943

Developers using C++ AMP will use a form of `parallel_for_each()` to launch data-parallel computations on accelerators. The behavior of `parallel_for_each` is similar to that of `std::for_each`: execute a function for each element in a range. The C++ AMP specialization over ranges of type `extent` and `tiled_extent` allow execution of functions on accelerators.

2947

The `parallel_for_each` function takes the following general forms:

2949

1. Non-tiled:

```
template <int N, typename Kernel>
void parallel_for_each(const extent<N>& compute_domain, const Kernel& f);
```

2953

2. Tiled:

```
template <int D0, int D1, int D2, typename Kernel>
void parallel_for_each(const tiled_extent<D0,D1,D2>& compute_domain, const Kernel& f);
```

2957

```
template <int D0, int D1, typename Kernel>
void parallel_for_each(const tiled_extent<D0,D1>& compute_domain, const Kernel& f);
```

2960

```
template <int D0, typename Kernel>
void parallel_for_each(const tiled_extent<D0>& compute_domain, const Kernel& f);
```

2963

A `parallel_for_each` invocation may be explicitly requested on a specific accelerator view

2965

1. Non-tiled:

```

2967     template <int N, typename Kernel>
2968         void parallel_for_each(const accelerator_view& accl_view,
2969                             const extent<N>& compute_domain, const Kernel& f);
2970
2971     2. Tiled:
2972         template <int D0, int D1, int D2, typename Kernel>
2973             void parallel_for_each(const accelerator_view& accl_view,
2974                                     const tiled_extent<D0,D1,D2>& compute_domain, const Kernel& f);
2975
2976         template <int D0, int D1, typename Kernel>
2977             void parallel_for_each(const accelerator_view& accl_view,
2978                                     const tiled_extent<D0,D1>& compute_domain, const Kernel& f);
2979
2980         template <int D0, typename Kernel>
2981             void parallel_for_each(const accelerator_view& accl_view,
2982                                     const tiled_extent<D0>& compute_domain, const Kernel& f);
2983
2984 A parallel_for_each over an extent represents a dense loop nest of independent serial loops.
2985
2986 When parallel_for_each executes, a parallel activity is spawned for each index in the compute domain. Each parallel
2987 activity is associated with an index value. (This index is an index<N> in the case of a non-tiled parallel_for_each, or a
2988 tiled_index<D0,D1,D2> in the case of a tiled parallel_for_each.) A parallel activity typically uses its index to access the
2989 appropriate locations in the input/output arrays.
2990
2991 A call to parallel_for_each behaves as if it were synchronous. In practice, the call may be asynchronous because it executes
2992 on a separate device, but since data copy-out is a synchronizing event, the developer cannot tell the difference.
2993
2994 There are no guarantees on the order and concurrency of the parallel activities spawned by the non-tiled parallel_for_each.
2995 Thus it is not valid to assume that one activity can wait for another sibling activity to complete for itself to make progress.
2996 This is discussed in further detail in section 3.
2997
2998 The tiled version of parallel_for_each organizes the parallel activities into fixed-size tiles of 1, 2, or 3 dimensions, as given by
2999 the tiled_extent<> argument. The tiled_extent provided as the first parameter to parallel_for_each must be divisible, along
3000 each of its dimensions, by the respective tile extent. Tiling beyond 3 dimensions is not supported. Threads (parallel
3001 activities) in the same tile have access to shared tile_static memory, and can use tiled_index::barrier.wait (4.5.3) to
3002 synchronize access to it.
3003
3004 When launching an amp-restricted kernel, the implementation of tiled parallel_for_each will provide the following
3005 minimum capabilities:
3006
3007     • The maximum number of tiles per dimension will be no less than 65535.
3008     • The maximum number of threads in a tile will be no less than 1024.
3009         ○ In 3D tiling, the maximal value of D0 will be no less than 64.
3010
3011 Microsoft-specific: When launching an amp-restricted kernel, the tiled parallel_for_each provides the above portable
3012 guarantees and no more. i.e.,
3013     • The maximum number of tiles per dimension is 65535.
3014         ○ In 3D tiling, the maximum value supported for D0 is 64.
3015
3016 The execution behind the parallel_for_each occurs on a certain accelerator, in the context of a certain accelerator view.
3017 This accelerator view may be passed explicitly to parallel_for_each (as an optional first argument). Otherwise, the target
3018 accelerator and the view using which work is submitted to the accelerator, is chosen from the objects of type array<T,N>
3019 and texture<T> that were captured in the kernel lambda. An implementation may require that all arrays and textures

```

3018 captured in the lambda must be on the same accelerator view; if not, an implementation is allowed to throw an exception. An  
 3019 implementation may also arrange for the specified data to be accessible on the selected accelerator view, rather than reject  
 3020 the call.  
 3021

**Microsoft-specific:** the Microsoft implementation of C++ AMP requires that all array and texture objects are co-located on the same accelerator view which is used, implicitly or explicitly in a [parallel\\_for\\_each](#) call.

3022 If the parallel\_for\_each kernel functor does not capture an array/texture object and neither is the target accelerator\_view  
 3023 for the kernel's execution is explicitly specified, the runtime is allowed to execute the kernel on any accelerator\_view on  
 3024 the default accelerator.  
 3025

3026

**Microsoft-specific:** In such a scenario, the Microsoft implementation of C++ AMP selects the target accelerator\_view for executing the parallel\_for\_each kernel as follows:

3027

- a. Determine the set of accelerator\_views where all array\_views referenced in the p\_f\_e kernel have cached copies
- b. From the above set, filter out any accelerator\_views that are not on the default accelerator. Additionally filter out accelerator\_views that do not have the capabilities required by the p\_f\_e kernel (debug intrinsics, number of UAVs)
- c. The default accelerator\_view of the default accelerator is selected as the target, if the resultant set from b. is empty, or contains that accelerator\_view
- d. Otherwise, any accelerator\_view from the resultant set from b., is arbitrarily selected as the target

3028

3029 The argument *f* of template-argument type *Kernel* to the [parallel\\_for\\_each](#) function must be a lambda or functor offering  
 3030 an appropriate function call operator which the implementation of [parallel\\_for\\_each](#) invokes with the instantiated index  
 3031 type. To execute on an accelerator, the function call operator must be marked *restrict(amp)* (but may have additional  
 3032 restrictions), and it must be callable from a caller passing in the instantiated index type. Overload resolution is handled as if  
 3033 the caller contained this code:  
 3034

```
3035 template <typename IndexType, typename Kernel>
3036 void parallel_for_each_stub(IndexType i, const Kernel& f) restrict(amp)
3037 {
3038     f(i);
3039 }
```

3040 Where the *Kernel f* argument is the same one passed into [parallel\\_for\\_each](#) by the caller, and the index instance *i* is the  
 3041 thread identifier, where *IndexType* is the following type:  
 3042

- Non-Tiled [parallel\\_for\\_each](#): *index<N>*, where *N* must be the same rank as the *extent<N>* used in the [parallel\\_for\\_each](#).
- Tiled [parallel\\_for\\_each](#): *tiled\_index<D0 [, D1 [, D2]]>*, where the tile extents must match those of the *tiled\_extent* used in the [parallel\\_for\\_each](#).

3043 The *tiled\_index<>* argument passed to the kernel contains a collection of indices including those that are relative to the  
 3044 current tile.  
 3045

3046

3047 The value returned by the kernel function, if any, is ignored.  
 3048

3049

**Microsoft-specific:**

3066     *In the Microsoft implementation of C++ AMP, every function that is referenced directly or indirectly by the kernel function, as  
 3067     well as the kernel function itself, must be inlineable<sup>4</sup>.*

## 3068     **7.1 Capturing Data in the Kernel Function Object**

3069     Since the kernel function object does not take any other arguments, all other data operated on by the kernel, other than  
 3070     the thread index, must be captured in the lambda or function object passed to [parallel\\_for\\_each](#). The function object shall  
 3071     be any amp-compatible class, struct or union type, including those introduced by lambda expressions.

## 3072     **7.2 Exception Behaviour**

3073     If an error occurs trying to launch the [parallel\\_for\\_each](#), an exception will be thrown. Exceptions can be thrown for the  
 3074     following reasons:

- 3075       1. Invalid extent passed
- 3076       2. (Optional) Not all arrays and/or textures reside on the accelerator\_view selected for execution
- 3077       3. Kernel using features not supported on the target accelerator
- 3078       4. Internal failure to allocate resources or to start the execution

## 3079     **8 Correctly Synchronized C++ AMP Programs**

3080     Correctly synchronized C++ AMP programs are correctly synchronized C++ programs which also adhere to a few additional  
 3081     C++ AMP rules, as follows:

- 3082       1. Accelerator-side execution
  - 3083           a. Concurrency rules for arbitrary sibling threads launched by a [parallel\\_for\\_each](#) call.
  - 3084           b. Semantics and correctness of tile barriers.
  - 3085           c. Semantics of atomic and memory fence operations.
- 3086       2. Host-side execution
  - 3087           a. Concurrency of accesses to C++ AMP containers between host-side operations: [copy](#), [synchronize](#),  
             [parallel\\_for\\_each](#) and the application of the various subscript operators of arrays and array views on the  
             host.
  - 3089           b. Accessing [arrays](#) or [array\\_view](#) data on the host.

### 3091     **8.1 Concurrency of sibling threads launched by a parallel\_for\_each call**

3092     In this section we will consider the relationship between sibling threads in a single [parallel\\_for\\_each](#) call. Interaction  
 3093     between separate [parallel\\_for\\_each](#) calls, copy operations and other host-side operations will be considered in the  
 3094     following sub-sections.

3095     A [parallel\\_for\\_each](#) call logically initiates the operation of multiple sibling threads, one for each coordinate in the [extent](#) or  
 3096     [tiled\\_extent](#) passed to it.

3097     All the threads launched by a [parallel\\_for\\_each](#) are potentially concurrent. Unless barriers are used, an implementation is  
 3098     free to schedule these threads in any order. In addition, the memory model for normal memory accesses is weak, that is  
 3099     operations could be arbitrarily reordered as long as each thread perceives to execute in its original program order. Thus any  
 3100     two memory operations from any two threads in a [parallel\\_for\\_each](#) are by default concurrent, unless the application has  
 3101     explicitly enforced an order between these two operations using atomic operations, fences or barriers.

3102     Conversely, an implementation may also schedule only a single logical thread at a time, in a non-cooperative manner, i.e.,  
 3103     without letting any other threads make any progress, with the exception of hitting a tile barrier or terminating. When a  
 3104     thread encounters a tile barrier, an implementation must wrest control from that thread and provide progress to some

---

<sup>4</sup> An implementation can employ whole-program compilation (such as link-time code-gen) to achieve this.

3108 other thread in the tile until they all have reached the barrier. Similarly, when a thread finishes execution, the system is  
 3109 obligated to execute steps from some other thread. Thus an implementation is obligated to switch context between  
 3110 threads only when a thread has hit a barrier (barriers pertain just to the tiled [parallel\\_for\\_each](#)), or is finished. An  
 3111 implementation doesn't have to admit any concurrency at a finer level than that which is dictated by barriers and thread  
 3112 termination. All implementations, however, are obligated to ensure progress is continually made, until all threads launched  
 3113 by a [parallel\\_for\\_each](#) are completed.

3114

3115 An immediate corollary is that C++ AMP doesn't provide a mechanism using which a thread could, without using tile  
 3116 barriers, poll for a change which needs to be effected by another thread. In particular, C++ AMP doesn't support locks  
 3117 which are implemented using atomic operations and fences, since a thread could end up polling forever, waiting for a lock  
 3118 to become available. The usage of tile barriers allows for creating a limited form of locking scoped to a thread tile. For  
 3119 example:

```
3120
3121 void tile_lock_example()
3122 {
3123     parallel for each(
3124         extent<1>(TILE_SIZE).tile<TILE_SIZE>(),
3125         [] (tiled_index<TILE_SIZE> tidx) restrict(amp)
3126     {
3127         tile_static int lock;
3128
3129         // Initialize lock:
3130         if (tidx.local[0] == 0) lock = 0;
3131         tidx.barrier.wait();
3132
3133         bool performed_my_exclusive_work = false;
3134         for (;;) {
3135             // try to acquire the lock
3136             if (!performed_my_exclusive_work && atomic_compare_exchange(&lock, 0, 1)) {
3137                 // The lock has been acquired - mutual exclusion from the rest of the threads in the tile
3138                 // is provided here....
3139                 some_synchronized_op();
3140
3141                 // Release the lock
3142                 atomic_exchange(&lock, 0);
3143                 performed_my_exclusive_work = true;
3144             }
3145             else {
3146                 // The lock wasn't acquired, or we are already finished. Perhaps we can do something
3147                 // else in the meanwhile.
3148                 some_non_exclusive_op();
3149             }
3150
3151             // The tile barrier ensures progress, so threads can spin in the for loop until they
3152             // are successful in acquiring the lock.
3153             tidx.barrier.wait();
3154         }
3155     });
3156 }
```

3157

**Informative:** More often than not, such non-deterministic locking within a tile is not really necessary, since a static schedule of the threads based on integer thread ID's is possible and results in more efficient and more maintainable code, but we bring this example here for completeness and to illustrate a valid form of polling.

3161

### 8.1.1 Correct usage of tile barriers

3162 Correct C++ AMP programs require all threads in a tile to hit all tile barriers uniformly. That is, at a minimum, when a  
 3163 thread encounters a particular [tile\\_barrier::wait](#) call site (or any other barrier method of class [tile\\_barrier](#)), all other threads  
 3164 in the tile must encounter the same call site.

3165

**Informative:** This requirement, however, is typically not sufficient in order to allow for efficient implementations. For  
 3166 example, it allows for the call stack of threads to differ, when they hit a barrier. In order to be able to generate good quality  
 3167 code for vector targets, much stronger constraints should be placed on the usage of barriers, as explained below.

3169  
 3170 C++ AMP requires all *active control flow expressions* leading to a tile barrier to be *tile-uniform*. Active control flow  
 3171 expressions are those guarding the scopes of all control flow constructs and logical expressions, which are actively being  
 3172 executed at a time a barrier is called. For example, the condition of an *if* statement is an active control flow expression as  
 3173 long as either the true or false hands of the *if* statement are still executing. If either of those hands contains a tile barrier,  
 3174 or leads to one through an arbitrary nesting of scopes and function calls, then the control flow expression controlling the *if*  
 3175 statement must be *tile-uniform*. What follows is an exhaustive list of control flow constructs which may lead to a barrier  
 3176 and their corresponding control expressions:

3177     *if (<control-expression>) <statement> else <statement>*  
 3178     *switch (<control-expression> { <cases> }*  
 3179     *for (<init-expression>; <control-expression>; <iteration-expression>) <statement>*  
 3180     *while (<control-expression>) <statement>*  
 3181     *do <statement> while(<control-expression>);*  
 3182     *<control-expression> ? <expression> : <expression>*  
 3183     *<control-expression> && <expression>*  
 3184     *<control-expression> || <expression>*  
 3185  
 3186

3187 All active control flow constructs are strictly nested in accordance to the program's text, starting from the scope of the  
 3188 lambda at the *parallel\_for\_each* all the way to the scope containing the barrier.

3189  
 3190 C++ AMP requires that, when a barrier is encountered by one thread:

- 3191     1. That the same barrier will be encountered by all other threads in the tile.
- 3192     2. That the sequence of active control flow statements and/or expressions be identical for all threads when they  
       reach the barrier.
- 3193     3. That each of the corresponding control expressions be *tile-uniform* (which is defined below).
- 3194     4. That any active control flow statement or expression hasn't been departed (necessarily in a non-uniform fashion)  
       by a *break*, *continue* or *return* statement. That is, any breaking statement which instructs the program to leave an  
       active scope must in itself behave as if it was a barrier, i.e., adhere to these preceding rules.

3198 Informally, a *tile-uniform expression* is an expression only involving variables, literals and function calls which have a  
 3199 uniform value throughout the tile. Formally, C++ AMP specifies that:

- 3201     5. *Tile-uniform* expressions may reference literals and template parameters
- 3202     6. *Tile-uniform* expressions may reference *const* (or effectively *const*) data members of the function object parameter  
       of *parallel\_for\_each*
- 3203     7. *Tile-uniform* expressions may reference *tiled\_index<,,>::tile*
- 3204     8. *Tile-uniform* expressions may reference values loaded from *tile\_static* variables as long as those values are loaded  
       immediately and uniformly after a tile barrier. That is, if the barrier and the load of the value occur at the same  
       function and the barrier dominates the load and no potential store into the same *tile\_static* variable intervenes  
       between the barrier and the load, then the loaded value will be considered *tile-uniform*
- 3205     9. Control expressions may reference *tile-uniform local variables and parameters*. Uniform local variables and  
       parameters are variables and parameters which are always initialized and assigned-to under uniform control flow  
       (that is, using the same rules which are defined here for barriers) and which are only assigned *tile-uniform*  
       expressions
- 3206     10. *Tile-uniform* expressions may reference the return values of functions which return *tile-uniform* expressions
- 3207     11. *Tile-uniform* expressions may not reference any expression not explicitly listed by the previous rules

3215  
 3216 An implementation is not obligated to warn when a barrier does not meet the criteria set forth above. An implementation  
 3217 may disqualify the compilation of programs which contain incorrect barrier usage. Conversely, an implementation may  
 3218 accept programs containing incorrect barrier usage and may execute them with undefined behavior.

3219   **8.1.2 Establishing order between operations of concurrent parallel\_for\_each threads**

3220   Threads may employ atomic operations, barriers and fences to establish a happens-before relationship encompassing their  
 3221   cumulative execution. When considering the correctness of the synchronization of programs, the following three aspects of  
 3222   the programs are relevant:

- 3223   1. The types of memory which are potentially accessed concurrently by different threads. The memory type can be:  
 3224       a. Global memory  
 3225       b. Tile-static memory
- 3226   2. The relationship between the threads which could potentially access the same piece of memory. They could be:  
 3227       a. Within the same thread tile  
 3228       b. Within separate threads tiles or sibling threads in the basic (non-tiled) parallel\_for\_each model.
- 3229   3. Memory operations which the program contains:  
 3230       a. Normal memory reads and writes.  
 3231       b. Atomic read-modify-write operations.  
 3232       c. Memory fences and barriers

3233   Informally, the C++ AMP memory model is a weak memory model consistent with the C++ memory model, with the  
 3234   following exceptions:

- 3235   1. Atomic operations do not necessarily create a sequentially consistent subset of execution. Atomic operations are  
 3236      only coherent, not sequentially consistent. That is, there doesn't necessarily exist a global linear order containing  
 3237      all atomic operations affecting all memory locations which were subjects of such operations. Rather, a separate  
 3238      global order exists for each memory location, and these per-location memory orders are not necessarily  
 3239      combinable into a single global order. (Note: this means an atomic operation does not constitute a memory fence.)
- 3240   2. Memory fence operations are limited in their effects to the thread tile they are performed within. When a thread  
 3241      from tile A executes a fence, the fence operation doesn't necessarily affect any other thread from any tile other  
 3242      than A.
- 3243   3. As a result of (1) and (2), the only mechanism available for cross-tile communication is atomic operations, and  
 3244      even when atomic operations are concerned, a linear order is only guaranteed to exist on a per-location basis, but  
 3245      not necessarily globally.
- 3246   4. Fences are bi-directional, meaning they have both acquire and release semantics.
- 3247   5. Fences can also be further scoped to a particular memory type (global vs. tile-static).
- 3248   6. Applying normal stores and atomic operations concurrently to the same memory location results in undefined  
 3249      behavior.
- 3250   7. Applying a normal load and an atomic operation concurrently to the same memory location is allowed (i.e., results  
 3251      in defined bavior).

3252   We will now provide a more formal characterization of the different categories of programs based on their adherence to  
 3253   synchronization rules. The three classes of adherence are

- 3254   1. *barrier-incorrect* programs,
- 3255   2. *racy* programs, and,
- 3256   3. *correctly-synchronized* programs.

3257   **8.1.2.1 Barrier-incorrect programs**

3258   A *barrier-incorrect* program is a program which doesn't adhere to the correct barrier usage rules specified in the previous  
 3259   section. Such programs always have undefined behavior. The remainder of this section discusses barrier-correct programs  
 3260   only.

3261   **8.1.2.2 Compatible memory operations**

3262   The following definition is later used in the definition of racy programs.

3264 Two memory operations applied to the same (or overlapping) memory location are *compatible* if they are both aligned and  
 3265 have the same data width, and either both operations are reads, or both operation are atomic, or one operation is a read  
 3266 and the other is atomic.

3267  
 3268 This is summarized by the following table in which  $T_1$  is a thread executing  $Op_1$  and  $T_2$  is a thread executing operation  $Op_2$ .  
 3269

| $Op_1$ | $Op_2$ | Compatible? |
|--------|--------|-------------|
| Atomic | Atomic | Yes         |
| Read   | Read   | Yes         |
| Read   | Atomic | Yes         |
| Write  | Any    | No          |

3270

### 3271 8.1.2.3 Concurrent memory operations

3272 The following definition is later used in the definition of racy programs.  
 3273

3274 Informally, two memory operations by different threads are considered *concurrent* if no order has been established  
 3275 between them. Order can be established between two memory operations only when they are executed by threads within  
 3276 the same tile. Thus any two memory operations by threads from different tiles are always concurrent, even if they are  
 3277 atomic. Within the same tile, order is established using fences and barriers. Barriers are a strong form of a fence.  
 3278

3279 Formally, Let  $\{T_1, \dots, T_N\}$  be the threads of a tile. Fix a sharable memory type (be it global or tile-static). Let  $M$  be the total set  
 3280 of memory operations of the given memory type performed by the collective of the threads in the tile.  
 3281

3282 Let  $F = \langle F_1, \dots, F_L \rangle$  be the set of memory fence operations of the given memory type, performed by the collective of threads  
 3283 in the tile, and organized arbitrarily into an ordered sequence.  
 3284

3285 Let  $P$  be a partitioning of  $M$  into a sequence of subsets  $P = \langle M_0, \dots, M_L \rangle$ , organized into an ordered sequence in an arbitrary  
 3286 fashion.  
 3287

3288 Let  $S$  be the interleaving of  $F$  and  $P$ ,  $S = \langle M_0, F_1, M_1, \dots, F_L, M_L \rangle$   
 3289

3290  $S$  is *conforming* if both of these conditions hold:

- 3291 1. **Adherence to program order:** For each  $T_i$ ,  $S$  respects the fences performed<sup>5</sup> by  $T_i$ . That is any operation performed  
 3292 by  $T_i$  before  $T_i$  performed fence  $F_j$  appears strictly before  $F_j$  in  $S$ , and similarly any operations performed by  $T_i$  after  
 3293  $F_j$  appears strictly after  $F_j$  in  $S$ .
- 3294 2. **Self-consistency:** For  $i < j$ , let  $M_i$  be a subset containing at least one store (atomic or non-atomic) into location  $L$  and  
 3295 let  $M_j$  be a subset containing at least a single load of  $L$ , and no stores into  $L$ . Further assume that no subset in-  
 3296 between  $M_i$  and  $M_j$  stores into  $L$ . Then  $S$  provides that all loads in  $M_j$  shall:
  - 3297 a. Return values stored into  $L$  by operations in  $M_i$ , and
  - 3298 b. For each thread  $T_i$ , the subset of  $T_i$  operations in  $M_j$  reading  $L$  shall all return the same value (which is  
 3299 necessarily one stored by an operation in  $M_i$ , as specified by condition (a) above).
- 3300 3. **Respecting initial values.** Let  $M_j$  be a subset containing a load of  $L$ , and no stores into  $L$ . Further assume that there  
 3301 is no  $M_i$  where  $i < j$  such that  $M_i$  contains a store into  $L$ . Then all loads of  $L$  in  $M_j$  will return the initial value of  $L$ .

3302 In such a conforming sequence  $S$ , two operations are *concurrent* if they have been executed by different threads and they  
 3303 belong to some common subset  $M_i$ . Two operations are *concurrent in an execution history* of a tile, if there exists a

---

<sup>5</sup> Here, performance of memory operations is assumed to strictly follow program order.

3304 conforming interleaving S as described herein in which the operations are concurrent. Two operations of a program are  
 3305 *concurrent* if there possibly exists an execution of the program in which they are concurrent.  
 3306

3307 A barrier behaves like a fence to establish order between operations, except it provides additional guarantees on the order  
 3308 of execution. Based on the above definition, a barrier is like a fence that only permits a certain kind of interleaving.  
 3309 Specifically, one in which the sequence of fences (F in the above formalization) has the fences , corresponding to the barrier  
 3310 execution by individual threads, appearing uninterrupted in S, without any memory operations interleaved between them.  
 3311 For example, consider the following program:

3312     C1  
 3313     Barrier  
 3314     C2  
 3315

3316 Assume that C1 and C2 are arbitrary sequences of code. Assume this program is executed by two threads T1 and T2, then  
 3317 the only possible conforming interleavings are given by the following pattern:  
 3318

3319     T1(C1) || T2(C1)  
 3320     T1(Barrier) || T2(Barrier)  
 3321     T1(C2) || T2(C2)  
 3322

3323 Where the || operator implies arbitrary interleaving of the two operand sequences.

#### 3324     8.1.2.4 Racy programs

3325 *Racy programs* are programs which have possible executions where at least two operations performed by two separate  
 3326 threads are both (a) incompatible AND (b) concurrent.  
 3327

3328 Racy programs do not have semantics assigned to them. They have undefined behavior.

#### 3329     8.1.2.5 Race-free programs

3330 Race-free programs are, simply, programs that are not racy. Race-free programs have the following semantics assigned to  
 3331 them:  
 3332

- 3333     1. If two memory operations are ordered (i.e., not concurrent) by fences and/or barriers, then the values  
 3334       loaded/stored will respect such an ordering.
- 3335     2. If two memory operations are concurrent then they must be atomic and/or reads performed by threads within the  
 3336       same tile. For each memory location X there exists an eventual total order including all such operations concurrent  
 3337       operations applied to X and obeying the semantics of loads and atomic read-modify-write transactions.

## 3338     8.2 Cumulative effects of a `parallel_for_each` call

3339 An invocation of `parallel_for_each` receives a function object, the contents of which are made available on the device. The  
 3340 function object may contain: `concurrency::array` reference data members, `concurrency::array_view` value data members,  
 3341 `concurrency::graphics::texture` reference data members, and `concurrency::graphics::writeonly_texture_view` value data  
 3342 members. (In addition, the function object may also contain additional, user defined data members.) Each of these  
 3343 members of the types `array`, `array_view`, `texture` and `write_only_texture_view`, could be constrained in the type of access it  
 3344 provides to kernel code. For example an `array<int,2>&` member provides both read and write access to the array, while a  
 3345 `const array<int,2>&` member provides just read access to the array. Similarly, an `array_view<int,2>` member provides read  
 3346 and write access, while an `array_view<const int,2>` member provides read access only.  
 3347

3348 The C++ AMP specification permits implementations in which the memory backing an `array`, `array_view` or `texture` could be  
 3349 shared between different accelerators, and possibly also the host, while also permitting implementations where data has to  
 3350 be copied, by the implementation, between different memory regions in order to support access by some hardware.  
 3351 Simulating coherence at a very granular level is too expensive in the case disjoint memory regions are required by the  
 3352 hardware. Therefore, in order to support both styles of implementation, this specification stipulates that `parallel_for_each`

3353 has the freedom to implement coherence over *array*, *array\_view*, and *texture* using coarse copying. Specifically, while a  
 3354 *parallel\_for\_each* call is being evaluated, implementations may:

- 3355 1. Load and/or store any location, in any order, any number of times, of each container which is passed into  
 3356 *parallel\_for\_each* in read/write mode.
- 3357 2. Load from any location, in any order, any number of times, of each container which is passed into  
 3358 *parallel\_for\_each* in read-only mode.

3359  
 3360 A *parallel\_for\_each* always behaves synchronously. That is, any observable side effects caused by any thread executing  
 3361 within a *parallel\_for\_each* call, or any side effects further affected by the implementation, due to the freedom it has in  
 3362 moving memory around, as stipulated above, shall be visible by the time *parallel\_for\_each* return.

3363 However, since the effects of *parallel\_for\_each* are constrained to changing values within *arrays*, *array\_views* and *textures*  
 3364 and each of these objects can synchronize its contents lazily upon access, an asynchronous implementation of  
 3365 *parallel\_for\_each* is possible, and encouraged. Nonetheless, implementations should still honor calls to  
 3366 *accelerator\_view::wait* by blocking until all lazily queued side-effects have been fully performed. Similarly, an  
 3367 implementation should ensure that all lazily queued side-effects preceding an *accelerator\_view::create\_marker* call have  
 3368 been fully performed before the *completion\_future* object which is retuned by *create\_marker* is made ready.

3369  
 3370  
 3371 **Informative:** Future versions of *parallel\_for\_each* may be less constrained in the changes they may affect to shared memory,  
 3372 and at that point an asynchronous implementation will no longer be valid. At that point, an explicitly asynchronous  
 3373 *parallel\_for\_each\_async* will be added to the specification.

3374 Even though an implementation could be coarse in the way it implements coherence, it still must provide true aliasing for  
 3375 *array\_views* which refer to the same home location. For example, assuming that *a1* and *a2* are both *array\_views*  
 3376 constructed on top of a 100-wide one dimensional *array*, with *a1* referring to elements [0...10] of the *array* and *a2* referring  
 3377 to elements [10...20] of the same *array*. If both *a1* and *a2* are accessible on a *parallel\_for\_each* call, then accessing *a1* at  
 3378 position 10 is identical to accessing the view *a2* at position 0, since they both refer to the same location of the *array* they  
 3379 are providing a view over, namely position 10 in the original *array*. This rules holds whenever and wherever *a1* and *a2* are  
 3380 accessible simultaneously, i.e., on the host and in *parallel\_for\_each* calls.

3381  
 3382 Thus, for example, an implementation could clone an *array\_view* passed into a *parallel\_for\_each* in read-only mode, and  
 3383 pass the cloned data to the device. It can create the clone using any order of reads from the original. The implementation  
 3384 may read the original a multiple number of times, perhaps in order to implement load-balancing or reliability features.

3385  
 3386 Similarly, an implementation could copy back results from an internally cloned *array*, *array\_view* or *texture*, onto the  
 3387 original data. It may overwrite any data in the original container, and it can do so multiple times in the realization of a  
 3388 single *parallel\_for\_each* call.

3389  
 3390 When two or more overlapping array views are passed to a *parallel\_for\_each*, an implementation could create a temporary  
 3391 array corresponding to a section of the original container which contains at a minimum the union of the views necessary for  
 3392 the call. This temporary array will hold the clones of the overlapping *array\_views* while maintaining their aliasing  
 3393 requirements.

3394  
 3395 The guarantee regarding aliasing of *array\_views* is provided for views which share the same *home location*. The home  
 3396 location of an *array\_view* is defined thus:

- 3397 1. In the case of an *array\_view* that is ultimately derived from an *array*, the home location is the *array*.
- 3398 2. In the case of an *array\_view* that is ultimately derived from a host pointer, the home location is the original *array*  
 3399 view created using the pointer.

3400  
 3401 This means that two different *array\_views* which have both been created, independently, on top of the same memory  
 3402 region are not guaranteed to appear coherent. In fact, creating and using top-level *array\_views* on the same host storage is  
 3403 not supported. In order for such *array\_view* to appear coherent, they must have a common top-level *array\_view* ancestor

3405 which they both ultimately were derived from, and that top-level `array_view` must be the only one which is constructed on  
 3406 top of the memory it refers to.

3407 This is illustrated in the next example:

```

3409
3410 #include <assert.h>
3411 #include <amp.h>
3412
3413 using namespace concurrency;
3414
3415 void coherence_buggy()
3416 {
3417     int storage[10];
3418     array_view<int> av1(10, &storage[0]);
3419     array_view<int> av2(10, &storage[0]); // error: av2 is top-level and aliases av1
3420     array_view<int> av3(5, &storage[5]); // error: av3 is top-level and aliases av1, av2
3421
3422     parallel_for_each( extent<1>(1), [=] (index<1>) restrict(amp) { av3[2] = 15; });
3423     parallel_for_each( extent<1>(1), [=] (index<1>) restrict(amp) { av2[7] = 16; });
3424     parallel_for_each( extent<1>(1), [=] (index<1>) restrict(amp) { av1[7] = 17; });
3425
3426     assert(av1[7] == av2[7]); // undefined results
3427     assert(av1[7] == av3[2]); // undefined results
3428 }
3429
3430 void coherence_ok()
3431 {
3432     int storage[10];
3433     array_view<int> av1(10, &storage[0]);
3434     array_view<int> av2(av1); // OK
3435     array_view<int> av3(av1.section(5,5)); // OK
3436
3437     parallel_for_each( extent<1>(1), [=] (index<1>) restrict(amp) { av3[2] = 15; });
3438     parallel_for_each( extent<1>(1), [=] (index<1>) restrict(amp) { av2[7] = 16; });
3439     parallel_for_each( extent<1>(1), [=] (index<1>) restrict(amp) { av1[7] = 17; });
3440
3441     assert(av1[7] == av2[7]); // OK, never fails, both equal 17
3442     assert(av1[7] == av3[2]); // OK, never fails, both equal 17
3443 }
```

3444 An implementation is not obligated to report such programmer's errors.

### 3446 8.3 Effects of `copy` and `copy_async` operations

3447 Copy operations are offered on `array`, `array_view` and `texture`.

3448 Copy operations copy a source host buffer, `array`, `array_view` or a `texture` to a destination object which can also be one of  
 3449 these four varieties (except host buffer to host buffer, which is handled by `std::copy`). A `copy` operation will read all  
 3450 elements of its source. It may read each element multiple times and it may read elements in any order. It may employ  
 3451 memory load instructions that are either coarser or more granular than the width of the primitive data types in the  
 3452 container, but it is guaranteed to never read a memory location which is strictly outside of the source container.

3453 Similarly, `copy` will overwrite each and every element in its output range. It may do so multiple times and in any order and  
 3454 may coarsen or break apart individual store operations, but it is guaranteed to never write a memory location which is  
 3455 strictly outside of the target container.

3460 A synchronous copy operation extends from the time the function is called until it has returned. During this time, any  
 3461 source location may be read and any destination location may be written. An asynchronous copy extends from the time  
 3462 `copy_async` is called until the time the `std::future` returned is ready.

3463  
 3464 As always, it is the programmer's responsibility not to call functions which could result in a race. For example, this program  
 3465 is racy because the two copy operations are concurrent and `b` is written to by the first parallel activity while it is being  
 3466 updated by the second parallel activity.  
 3467

3468  
 3469     array<int> a(100), b(100), c(100);  
 3470     parallel\_invoke(  
 3471         [&] { copy(a,b); },  
 3472         [&] { copy(b,c); });  
 3473

## 3474 8.4 Effects of `array_view::synchronize`, `synchronize_async` and `refresh` functions

3475  
 3476 An `array_view` may be constructed to wrap over a host side pointer. For such `array_views`, it is generally forbidden to  
 3477 access the underlying `array_view` storage directly, as long as the `array_view` exists. Access to the storage area is generally  
 3478 accomplished indirectly through the `array_view`. However, `array_view` offers mechanisms to synchronize and refresh its  
 3479 contents, which do allow accessing the underlying memory directly. These mechanisms are described below.  
 3480

3481 Reading of the underlying storage is possible under the condition that the view has been first *synchronized* back to its home  
 3482 storage. This is performed using the `synchronize` or `synchronize_async` member functions of `array_view`.  
 3483

3484 When a top-level view is initially created on top of a raw buffer, it is synchronized with it. After it has been constructed, a  
 3485 top-level view, as well as derived views, may lose coherence with the underlying host-side raw memory buffer if the  
 3486 `array_view` is passed to `parallel_for_each` as a mutable view, or if the view is a target of a copy operation. In order to  
 3487 restore coherence with host-side underlying memory `synchronize` or `synchronize_async` must be called. Synchronization is  
 3488 restored when `synchronize` returns, or when the completion\_future returned by `synchronize_async` is ready.  
 3489

3490 For the sake of composition with `parallel_for_each`, `copy`, and all other host-side operations involving a view, `synchronize`  
 3491 should be considered a read of the entire data section referred to by the view, as if it was the source of a copy operation,  
 3492 and thus it must not be executed concurrently with any other operation involving writing the view. Note that even though  
 3493 `synchronize` does potentially modify the underlying host memory, it is logically a no-op as it doesn't affect the logical  
 3494 contents of the array. As such, it is allowed to execute concurrently with other operations which read the array view. As  
 3495 with `copy`, `synchronize` works at the granularity of the view it is applied to, e.g., synchronizing a view representing a sub-  
 3496 section of a parent view doesn't necessarily synchronize the entire parent view. It is just guaranteed to synchronize the  
 3497 overlapping portions of such related views.  
 3498

3499 `array_views` are also required to synchronize their home storage:

- 3500     1. Before they are destructed if and only if it is the last view of the underlying data container.  
 3501     2. When they are accessed using the subscript operator or the `.data()` method (on said home location)  
 3502

3503 As a result of (1), any errors in synchronization which may be encountered during destruction of arrays views will not be  
 3504 propagated through the destructor. Users are therefore encouraged to ensure that `array_views` which may contain  
 3505 unsynchronized data are explicitly synchronized before they are destructed.  
 3506

3507 As a result of (2), the implementation of the subscript operator may need to contain a coherence enforcing check,  
 3508 especially on platforms where the accelerator hardware and host memory are not shared, and therefore coherence is  
 3509 managed explicitly by the C++ AMP runtime. Such a check may be detrimental for code desiring to achieve high  
 3510 performance through vectorization of the array view accesses. Therefore it is recommended for such performance-

3511 sensitive code to obtain a pointer to the beginning of a “run” and perform the low-level accesses needed based off of the  
 3512 raw pointer into the `array_view`. `array_views` are guaranteed to be contiguous in the unit-stride dimension, which enables  
 3513 this style of coding. Furthermore, the code may explicitly synchronize the `array_view` and at that point read the home  
 3514 storage directly, without the mediation of the view.

3515  
 3516 Sometimes it is desirable to also allow refreshing of a view by directly from its underlying memory. The `refresh` member  
 3517 function is provided for this task. This function revokes any caches associated with the view and resynchronizes the view’s  
 3518 contents with the underlying memory. As such it may not be invoked concurrently with any other operation that accesses  
 3519 the view’s data. However, it is safe to assume that `refresh` doesn’t modify the view’s underlying data and therefore  
 3520 concurrent read access to the underlying data is allowed during `refresh`’s operation and after `refresh` has returned, till the  
 3521 point when coherence may have been lost again, as has been described above in the discussion on the `synchronize` member  
 3522 function.

## 3523 9 Math Functions

3524  
 3525 C++ AMP contains a rich library of floating point math functions that can be used in an accelerated computation. The C++  
 3526 AMP library comes in two flavors, each contained in a separate namespace. The functions contained in the  
 3527 `concurrency::fast_math` namespace support only single-precision (`float`) operands and are optimized for performance at the  
 3528 expense of accuracy. The functions contained in the `concurrency::precise_math` namespace support both single and double  
 3529 precision (`double`) operands and are optimized for accuracy at the expense of performance. The two namespaces cannot  
 3530 be used together without introducing ambiguities. The accuracy of the functions in the `concurrency::precise_math`  
 3531 namespace shall be at least as high as those in the `concurrency::fast_math` namespace.

3532 All functions are available in the `<amp_math.h>` header file, and all are decorated `restrict(amp)`.  
 3533

### 3535 9.1 fast\_math

3536 Functions in the `fast_math` namespace are designed for computations where accuracy is not a prime requirement, and  
 3537 therefore the minimum precision is implementation-defined.

3539 Not all functions available in `precise_math` are available in `fast_math`.  
 3541

| C++ API function                                                                          | Description                                                                                                                                                                                                                                                                            |
|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>float acosf(float x)</code><br><code>float acos(float x)</code>                     | Returns the arc cosine in radians and the value is mathematically defined to be between 0 and PI (inclusive).                                                                                                                                                                          |
| <code>float asinf(float x)</code><br><code>float asin(float x)</code>                     | Returns the arc sine in radians and the value is mathematically defined to be between -PI/2 and PI/2 (inclusive).                                                                                                                                                                      |
| <code>float atanf(float x)</code><br><code>float atan(float x)</code>                     | Returns the arc tangent in radians and the value is mathematically defined to be between -PI/2 and PI/2 (inclusive).                                                                                                                                                                   |
| <code>float atan2f(float y, float x)</code><br><code>float atan2(float y, float x)</code> | Calculates the arc tangent of the two variables x and y. It is similar to calculating the arc tangent of $y / x$ , except that the signs of both arguments are used to determine the quadrant of the result.). Returns the result in radians, which is between -PI and PI (inclusive). |
| <code>float ceilf(float x)</code><br><code>float ceil(float x)</code>                     | Rounds x up to the nearest integer.                                                                                                                                                                                                                                                    |
| <code>float cosf(float x)</code><br><code>float cos(float x)</code>                       | Returns the cosine of x.                                                                                                                                                                                                                                                               |
| <code>float coshf(float x)</code><br><code>float cosh(float x)</code>                     | Returns the hyperbolic cosine of x.                                                                                                                                                                                                                                                    |

|                                                                                                                 |                                                                                                                                                       |
|-----------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>float expf(float x)</code><br><code>float exp(float x)</code>                                             | Returns the value of e (the base of natural logarithms) raised to the power of x.                                                                     |
| <code>float exp2f(float x)</code><br><code>float exp2(float x)</code>                                           | Returns the value of 2 raised to the power of x.                                                                                                      |
| <code>float fabsf(float x)</code><br><code>float fabs(float x)</code>                                           | Returns the absolute value of floating-point number                                                                                                   |
| <code>float floorf(float x)</code><br><code>float floor(float x)</code>                                         | Rounds x down to the nearest integer.                                                                                                                 |
| <code>float fmaxf(float x, float y)</code><br><code>float fmax(float x, float y)</code>                         | Selects the greater of x and y.                                                                                                                       |
| <code>float fminf(float x, float y)</code><br><code>float fmin(float x, float y)</code>                         | Selects the lesser of x and y.                                                                                                                        |
| <code>float fmodf(float x, float y)</code><br><code>float fmod(float x, float y)</code>                         | Computes the remainder of dividing x by y. The return value is $x - n * y$ , where n is the quotient of $x / y$ , rounded towards zero to an integer. |
| <code>float frexpf(float x, int * exp)</code><br><code>float frexp(float x, int * exp)</code>                   | Splits the number x into a normalized fraction and an exponent which is stored in exp.                                                                |
| <code>int isfinite(float x)</code>                                                                              | Determines if x is finite.                                                                                                                            |
| <code>int isinf(float x)</code>                                                                                 | Determines if x is infinite.                                                                                                                          |
| <code>int isnan(float x)</code>                                                                                 | Determines if x is NAN.                                                                                                                               |
| <code>float ldexpf(float x, int exp)</code><br><code>float ldexp(float x, int exp)</code>                       | Returns the result of multiplying the floating-point number x by 2 raised to the power exp                                                            |
| <code>float logf(float x)</code><br><code>float log(float x)</code>                                             | Returns the natural logarithm of x.                                                                                                                   |
| <code>float log10f(float x)</code><br><code>float log10(float x)</code>                                         | Returns the base 10 logarithm of x.                                                                                                                   |
| <code>float log2f(float x)</code><br><code>float log2(float x)</code>                                           | Returns the base 2 logarithm of x.                                                                                                                    |
| <code>float modff(float x, float * iptr)</code><br><code>float modf(float x, float * iptr)</code>               | Breaks the argument x into an integral part and a fractional part, each of which has the same sign as x. The integral part is stored in iptr.         |
| <code>float powf(float x, float y)</code><br><code>float pow(float x, float y)</code>                           | Returns the value of x raised to the power of y.                                                                                                      |
| <code>float roundff(float x)</code><br><code>float round(float x)</code>                                        | Rounds x to the nearest integer.                                                                                                                      |
| <code>float rsqrtf(float x)</code><br><code>float rsqrt(float x)</code>                                         | Returns the reciprocal of the square root of x.                                                                                                       |
| <code>int signbitf(float x)</code><br><code>int signbit(float x)</code>                                         | Returns a non-zero value if the value of X has its sign bit set.                                                                                      |
| <code>float sinff(float x)</code><br><code>float sin(float x)</code>                                            | Returns the sine of x.                                                                                                                                |
| <code>void sincosf(float x, float* s, float* c)</code><br><code>void sincos(float x, float* s, float* c)</code> | Returns the sine and cosine of x.                                                                                                                     |
| <code>float sinhff(float x)</code><br><code>float sinh(float x)</code>                                          | Returns the hyperbolic sine of x.                                                                                                                     |
| <code>float sqrtf(float x)</code><br><code>float sqrt(float x)</code>                                           | Returns the non-negative square root of x                                                                                                             |
| <code>float tanff(float x)</code><br><code>float tan(float x)</code>                                            | Returns the tangent of x.                                                                                                                             |
| <code>float tanhf(float x)</code><br><code>float tanh(float x)</code>                                           | Returns the hyperbolic tangent of x.                                                                                                                  |

|                                               |                                                               |
|-----------------------------------------------|---------------------------------------------------------------|
| float truncf(float x)<br>float trunc(float x) | Rounds x to the nearest integer not larger in absolute value. |
|-----------------------------------------------|---------------------------------------------------------------|

3542  
 3543 The following list of standard math functions from the “`std::`” namespace shall be imported into the `concurrency::fast_math`  
 3544 namespace:

3545  
 3546     **using** `std::acosf;`  
 3547     **using** `std::asinf;`  
 3548     **using** `std::atanf;`  
 3549     **using** `std::atan2f;`  
 3550     **using** `std::ceilf;`  
 3551     **using** `std::cosf;`  
 3552     **using** `std::coshf;`  
 3553     **using** `std::expf;`  
 3554     **using** `std::exp2f;`  
 3555     **using** `std::fabsf;`  
 3556     **using** `std::floorf;`  
 3557     **using** `std::fmaxf;`  
 3558     **using** `std::fminf;`  
 3559     **using** `std::fmodf;`  
 3560     **using** `std::frexpf;`  
 3561     **using** `std::ldexpf;`  
 3562     **using** `std::logf;`  
 3563     **using** `std::log10f;`  
 3564     **using** `std::log2f;`  
 3565     **using** `std::modff;`  
 3566     **using** `std::powf;`  
 3567     **using** `std::roundf;`  
 3568     **using** `std::sinf;`  
 3569     **using** `std::sinhf;`  
 3570     **using** `std::sqrtf;`  
 3571     **using** `std::tanf;`  
 3572     **using** `std::tanhf;`  
 3573     **using** `std::truncf;`  
 3574  
 3575     **using** `std::acos;`  
 3576     **using** `std::asin;`  
 3577     **using** `std::atan;`  
 3578     **using** `std::atan2;`  
 3579     **using** `std::ceil;`  
 3580     **using** `std::cos;`  
 3581     **using** `std::cosh;`  
 3582     **using** `std::exp;`  
 3583     **using** `std::exp2;`  
 3584     **using** `std::fabs;`  
 3585     **using** `std::floor;`  
 3586     **using** `std::fmax;`  
 3587     **using** `std::fmin;`  
 3588     **using** `std::fmod;`  
 3589     **using** `std::frexp;`  
 3590     **using** `std::ldexp;`  
 3591     **using** `std::log;`  
 3592     **using** `std::log10;`  
 3593     **using** `std::log2;`  
 3594     **using** `std::modf;`  
 3595     **using** `std::pow;`  
 3596     **using** `std::round;`

```

3597     using std::sin;
3598     using std::sinh;
3599     using std::sqrt;
3600     using std::tan;
3601     using std::tanh;
3602     using std::trunc;
3603
3604 Importing these names into the fast_math namespace enables each of them to be called in unqualified syntax from a
3605 function that has both "restrict(cpu,amp)" restrictions. E.g.,
3606
3607 void compute() restrict(cpu,amp) {
3608     ...
3609     float x = cos(y); // resolves to std::cos in "cpu" context; else fast_math::cos in "amp" context
3610     ...
3611 }

```

## 3612     9.2 precise\_math

3613 Functions in the *precise\_math* namespace are designed for computations where accuracy is required. In the table below,
3614 the precision of each function is stated in units of "ulps" (error in last position).

3615 Functions in the *precise\_math* namespace also support both single and double precision, and are therefore dependent
3616 upon double-precision support in the underlying hardware, even for single-precision variants.

3618

| C++ API function                                                                                        | Description                                                                                                                                                                                                                                                                         | Precision<br>(float) | Precision<br>(double) |
|---------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-----------------------|
| float acosf(float x)<br><br>float acos(float x)<br>double acos(double x)                                | Returns the arc cosine in radians and the value is mathematically defined to be between 0 and PI (inclusive).                                                                                                                                                                       | 3                    | 2                     |
| float acoshf(float x)<br><br>float acosh(float x)<br>double acosh(double x)                             | Returns the hyperbolic arccosine.                                                                                                                                                                                                                                                   | 4                    | 2                     |
| float asinf(float x)<br><br>float asin(float x)<br>double asin(double x)                                | Returns the arc sine in radians and the value is mathematically defined to be between -PI/2 and PI/2 (inclusive).                                                                                                                                                                   | 4                    | 2                     |
| float asinhf(float x)<br><br>float asinh(float x)<br>double asinh(double x)                             | Returns the hyperbolic arcsine.                                                                                                                                                                                                                                                     | 3                    | 2                     |
| float atanf(float x)<br><br>float atan(float x)<br>double atan(double x)                                | Returns the arc tangent in radians and the value is mathematically defined to be between -PI/2 and PI/2 (inclusive).                                                                                                                                                                | 2                    | 2                     |
| float atanhf(float x)<br><br>float atanh(float x)<br>double atanh(double x)                             | Returns the hyperbolic arctangent.                                                                                                                                                                                                                                                  | 3                    | 2                     |
| float atan2f(float y, float x)<br><br>float atan2(float y, float x)<br>double atan2(double y, double x) | Calculates the arc tangent of the two variables x and y. It is similar to calculating the arc tangent of y / x, except that the signs of both arguments are used to determine the quadrant of the result.). Returns the result in radians, which is between -PI and PI (inclusive). | 3                    | 2                     |
| float cbrtf(float x)                                                                                    | Returns the (real) cube root of x.                                                                                                                                                                                                                                                  | 1                    | 1                     |

|                                                                                                                  |                                                                                                                                                    |     |     |
|------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|-----|-----|
| float cbrt(float x)<br>double cbrt(double x)                                                                     |                                                                                                                                                    |     |     |
| float ceilf(float x)<br><br>float ceil(float x)<br>double ceil(double x)                                         | Rounds x up to the nearest integer.                                                                                                                | 0   | 0   |
| float copysignf(float x, float y)<br><br>float copysign(float x, float y)<br>double copysign(double x, double y) | Return a value whose absolute value matches that of x, but whose sign matches that of y. If x is a NaN, then a NaN with the sign of y is returned. | N/A | N/A |
| float cosf(float x)<br><br>float cos(float x)<br>double cos(double x)                                            | Returns the cosine of x.                                                                                                                           | 2   | 2   |
| float coshf(float x)<br><br>float cosh(float x)<br>double cosh(double x)                                         | Returns the hyperbolic cosine of x.                                                                                                                | 2   | 2   |
| float cospi(float x)<br><br>float cospi(float x)<br>double cospi(double x)                                       | Returns the cosine of pi * x.                                                                                                                      | 2   | 2   |
| float erff(float x)<br><br>float erf(float x)<br>double erf(double x)                                            | Returns the error function of x; defined as<br>$\text{erf}(x) = 2/\sqrt{\pi} \int_0^x e^{-t^2} dt$                                                 | 3   | 2   |
| float erfcf(float x)<br><br>float erfc(float x)<br>double erfc(double x)                                         | Returns the complementary error function of x that is $1.0 - \text{erf}(x)$ .                                                                      | 6   | 5   |
| float erfinvf(float x)<br><br>float erfinv(float x)<br>double erfinv(double x)                                   | Returns the inverse error function.                                                                                                                | 3   | 8   |
| float erfcinvf(float x)<br><br>float erfcinv(float x)<br>double erfcinv(double x)                                | Returns the inverse of the complementary error function.                                                                                           | 7   | 8   |
| float expf(float x)<br><br>float exp(float x)<br>double exp(double x)                                            | Returns the value of e (the base of natural logarithms) raised to the power of x.                                                                  | 2   | 1   |
| float exp2f(float x)<br><br>float exp2(float x)<br>double exp2(double x)                                         | Returns the value of 2 raised to the power of x.                                                                                                   | 2   | 1   |
| float exp10f(float x)<br><br>float exp10(float x)<br>double exp10(double x)                                      | Returns the value of 10 raised to the power of x.                                                                                                  | 2   | 1   |
| float expm1f(float x)<br><br>float expm1(float x)<br>double expm1(double x)                                      | Returns a value equivalent to ' $\exp(x) - 1$ '                                                                                                    | 1   | 1   |
| float fabsf(float x)<br><br>float fabs(float x)<br>double fabs(double x)                                         | Returns the absolute value of floating-point number                                                                                                | N/A | N/A |

|                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
|----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|---------------------------|
| float fdimf(float x, float y)                                                    | These functions return $\max(x-y, 0)$ . If x or y or both are NaN, Nan is returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 0   | 0                         |
| float fdim(float x, float y)<br>double fdim(double x, double y)                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| float floorf(float x)                                                            | Rounds x down to the nearest integer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 0   | 0                         |
| float floor(float x)<br>double floor(double x)                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| float fmaf(float x, float y, float z)                                            | Computes $(x * y) + z$ , rounded as one ternary operation: they compute the value (as if) to infinite precision and round once to the result format, according to the current rounding mode. A range error may occur.                                                                                                                                                                                                                                                                                                                                                                                                   | 0   | <sup>0</sup> <sup>6</sup> |
| float fma(float x, float y, float z)<br>double fma(double x, double y, double z) |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| float fmaxf(float x, float y)                                                    | Selects the greater of x and y.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | N/A | N/A                       |
| float fmax(float x, float y)<br>double fmax(double x, double y)                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| float fminf(float x, float y)                                                    | Selects the lesser of x and y.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | N/A | N/A                       |
| float fmin(float x, float y)<br>double fmin(double x, double y)                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| float fmodf(float x, float y)                                                    | Computes the remainder of dividing x by y. The return value is $x - n * y$ , where n is the quotient of x / y, rounded towards zero to an integer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 0   | 0                         |
| float fmod(float x, float y)<br>double fmod(double x, double y)                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| int fpclassify(float x);<br><br>int fpclassify(double x);                        | Floating point numbers can have special values, such as infinite or NaN. With the macro fpclassify(x) you can find out what type x is. The function takes any floating-point expression as argument. The result is one of the following values: <ul style="list-style-type: none"> <li>• FP_NAN : x is "Not a Number".</li> <li>• FP_INFINITE: x is either plus or minus infinity.</li> <li>• FP_ZERO: x is zero.</li> <li>• FP_SUBNORMAL : x is too small to be represented in normalized format.</li> <li>• FP_NORMAL : if nothing of the above is correct then it must be a normal floating-point number.</li> </ul> | N/A | N/A                       |
| float frexpf(float x, int * exp)                                                 | Splits the number x into a normalized fraction and an exponent which is stored in exp.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0   | 0                         |
| float frexp(float x, int * exp)<br>double frexp(double x, int * exp)             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| float hypotf(float x, float y)                                                   | Returns $\sqrt{x^2 + y^2}$ . This is the length of the hypotenuse of a right-angle triangle with sides of length x and y, or the distance of the point (x,y) from the origin.                                                                                                                                                                                                                                                                                                                                                                                                                                           | 3   | 2                         |
| float hypot(float x, float y)<br>double hypot(double x, double y)                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| int ilogbf (float x)                                                             | Return the exponent part of their argument as a signed integer. When no error occurs, these functions are equivalent to the corresponding logb() functions, cast to (int). An error will occur for zero and infinity and NaN, and possibly for overflow.                                                                                                                                                                                                                                                                                                                                                                | 0   | 0                         |
| int ilogb(float x)<br>int ilogb(double x)                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| int isfinite(float x)                                                            | Determines if x is finite.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | N/A | N/A                       |
| int isfinite(double x)                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |
| int isninf(float x)                                                              | Determines if x is infinite.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | N/A | N/A                       |
| int isninf(double x)                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |                           |

<sup>6</sup> IEEE-754 round to nearest even.

|                                                                            |                                                                                                                                                                                                                                                                                                                                  |                |                |
|----------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|----------------|
| int isnan(float x)                                                         | Determines if x is NAN.                                                                                                                                                                                                                                                                                                          | N/A            | N/A            |
| int isnan(double x)                                                        |                                                                                                                                                                                                                                                                                                                                  |                |                |
| int isnormal(float x)                                                      | Determines if x is normal.                                                                                                                                                                                                                                                                                                       | N/A            | N/A            |
| int isnormal(double x)                                                     |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float ldexpf(float x, int exp)                                             | Returns the result of multiplying the floating-point number x by 2 raised to the power exp                                                                                                                                                                                                                                       | 0              | 0              |
| float ldexp(float x, int exp)<br>double ldexpf(double x, int exp)          |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float lgammaf(float x, int * sign)                                         | Computes the natural logarithm of the absolute value of gamma of x. A range error occurs if x is too large. A range error may occur if x is a negative integer or zero. Stores the sign of the gamma function of x in parameter sign.                                                                                            | 6 <sup>7</sup> | 4 <sup>8</sup> |
| float lgamma(float x, int * sign)<br>double lgamma(double x, int * sign)   |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float logf(float x)                                                        | Returns the natural logarithm of x.                                                                                                                                                                                                                                                                                              | 1              | 1              |
| float log(float x)<br>double log(double x)                                 |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float log10f(float x)                                                      | Returns the base 10 logarithm of x.                                                                                                                                                                                                                                                                                              | 3              | 1              |
| float log10(float x)<br>double log10(double x)                             |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float log2f(float x)                                                       | Returns the base 2 logarithm of x.                                                                                                                                                                                                                                                                                               | 3              | 1              |
| float log2(float x)<br>double log2(double x)                               |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float log1pf (float x)                                                     | Returns a value equivalent to 'log (1 + x)'. It is computed in a way that is accurate even if the value of x is near zero.                                                                                                                                                                                                       | 2              | 1              |
| float log1p(float x)<br>double log1p(double x)                             |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float logbf(float x)                                                       | These functions extract the exponent of x and return it as a floating-point value. If FLT_RADIX is two, logb(x) is equal to floor(log2(x)), except it's probably faster.                                                                                                                                                         | 0              | 0              |
| float logb(float x)<br>double logb(double x)                               | If x is de-normalized, logb() returns the exponent x would have if it were normalized.                                                                                                                                                                                                                                           |                |                |
| float modff(float x, float * iptr)                                         | Breaks the argument x into an integral part and a fractional part, each of which has the same sign as x. The integral part is stored in iptr.                                                                                                                                                                                    | 0              | 0              |
| float modf(float x, float * iptr)<br>double modff(double x, double * iptr) |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float nanf(int tagp)                                                       | return a representation (determined by tagp) of a quiet NaN. If the implementation does not support quiet NaNs, these functions return zero.                                                                                                                                                                                     | N/A            | N/A            |
| float nanf(int tagp)<br>double nan(int tagp)                               |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float nearbyintf(float x)                                                  | Rounds the argument to an integer value in floating point format, using the current rounding direction                                                                                                                                                                                                                           | 0              |                |
| float nearbyint(float x)<br>double nearbyint(double x)                     |                                                                                                                                                                                                                                                                                                                                  |                |                |
| float nextafterf(float x, float y)                                         | Returns the next representable neighbor of x in the direction towards y. The size of the step between x and the result depends on the type of the result. If x = y the function simply returns y. If either value is NaN, then NaN is returned. Otherwise a value corresponding to the value of the least significant bit in the | N/A            | N/A            |
| float nextafter(float x, float y)<br>double nextafter(double x, double y)  |                                                                                                                                                                                                                                                                                                                                  |                |                |

<sup>7</sup> Outside interval -10.001 ... -2.264; larger inside.<sup>8</sup> Outside interval -10.001 ... -2.264; larger inside.

|                                                                                                                                                |                                                                                                                                                                                                                                                                                                                       |     |                |
|------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|----------------|
|                                                                                                                                                |                                                                                                                                                                                                                                                                                                                       |     |                |
| float powf(float x, float y)<br><br>float pow(float x, float y)<br>double pow(double x, double y)                                              | mantissa is added or subtracted, depending on the direction.<br><br>Returns the value of x raised to the power of y.                                                                                                                                                                                                  | 8   | 2              |
| float rcbtf(float x)<br><br>float rcbt(float x)<br>double rcbt(double x)                                                                       | Calculates reciprocal of the (real) cube root of x                                                                                                                                                                                                                                                                    | 2   | 1              |
| float remainderf(float x, float y)<br><br>float remainder(float x, float y)<br>double remainder(double x, double y)                            | Computes the remainder of dividing x by y. The return value is $x - n * y$ , where n is the value $x / y$ , rounded to the nearest integer. If this quotient is $1/2 \pmod{1}$ , it is rounded to the nearest even number (independent of the current rounding mode). If the return value is 0, it has the sign of x. | 0   | 0              |
| float remquof(float x, float y, int * quo)<br><br>float remquo(float x, float y, int * quo)<br>double remquo(double x, double y, int * quo)    | Computes the remainder and part of the quotient upon division of x by y. A few bits of the quotient are stored via the quo pointer. The remainder is returned.                                                                                                                                                        | 0   | 0              |
| float roundf(float x)<br><br>float round(float x)<br>double round(double x)                                                                    | Rounds x to the nearest integer.                                                                                                                                                                                                                                                                                      | 0   | 0              |
| float rsqrtf(float x)<br><br>float rsqrt(float x)<br>double rsqrt(double x)                                                                    | Returns the reciprocal of the square root of x.                                                                                                                                                                                                                                                                       | 2   | 1              |
| float sinpiif(float x)<br><br>float sinpi(float x)<br>double sinpi(double x)                                                                   | Returns the sine of $\pi * x$ .                                                                                                                                                                                                                                                                                       | 2   | 2              |
| float scalbf(float x, float exp)<br><br>float scalb(float x, float exp)<br>double scalb(double x, double exp)                                  | Multiplies their first argument x by FLT_RADIX (probably 2) to the power exp.                                                                                                                                                                                                                                         | 0   | 0              |
| float scalbnf(float x, int exp)<br><br>float scalbn(float x, int exp)<br>double scalbn(double x, int exp)                                      | Multiplies their first argument x by FLT_RADIX (probably 2) to the power exp. If FLT_RADIX equals 2, then scalbn() is equivalent to ldexp(). The value of FLT_RADIX is found in <float.h>.                                                                                                                            | 0   | 0              |
| int signbitf(float x)<br>int signbit(float x)<br>int signbit(double x)                                                                         | Returns a non-zero value if the value of X has its sign bit set.                                                                                                                                                                                                                                                      | N/A | N/A            |
| float sinf(float x)<br><br>float sin(float x)<br>double sin(double x)                                                                          | Returns the sine of x.                                                                                                                                                                                                                                                                                                | 2   | 2              |
| void sincosf(float x, float * s, float * c)<br><br>void sincos(float x, float * s, float * c)<br>void sincos(double x, double * s, double * c) | Returns the sine and cosine of x.                                                                                                                                                                                                                                                                                     | 2   | 2              |
| float sinhff(float x)<br><br>float sinh(float x)<br>double sinh(double x)                                                                      | Returns the hyperbolic sine of x.                                                                                                                                                                                                                                                                                     | 3   | 2              |
| float sqrtf(float x)                                                                                                                           | Returns the non-negative square root of x                                                                                                                                                                                                                                                                             | 0   | 0 <sup>9</sup> |

<sup>9</sup> IEEE-754 round to nearest even.

|                                                                                |                                                                           |    |   |
|--------------------------------------------------------------------------------|---------------------------------------------------------------------------|----|---|
| float sqrt(float x)<br>double sqrt(double x)                                   |                                                                           |    |   |
| float tgammaf(float x)<br><br>float tgamma(float x)<br>double tgamma(double x) | This function returns the value of the Gamma function for the argument x. | 11 | 8 |
| float tanf(float x)<br><br>float tan(float x)<br>double tan(double x)          | Returns the tangent of x.                                                 | 4  | 2 |
| float tanhf(float x)<br><br>float tanh(float x)<br>double tanh(double x)       | Returns the hyperbolic tangent of x.                                      | 2  | 2 |
| float tanpi(float x)<br><br>float tanpi(float x)<br>double tanpi(double x)     | Returns the tangent of pi * x.                                            | 2  | 2 |
| float truncf(float x)<br><br>float trunc(float x)<br>double trunc(double x)    | Rounds x to the nearest integer not larger in absolute value.             | 0  | 0 |

3619

3620 The following list of standard math functions from the “std::” namespace shall be imported into the  
 3621 concurrency::precise\_math namespace:

3622

```

3623     using std::acosf;
3624     using std::asinf;
3625     using std::atanf;
3626     using std::atan2f;
3627     using std::ceilf;
3628     using std::cosf;
3629     using std::coshf;
3630     using std::expf;
3631     using std::fabsf;
3632     using std::floorf;
3633     using std::fmodf;
3634     using std::frexpf;
3635     using std::hypotf;
3636     using std::ldexpf;
3637     using std::logf;
3638     using std::log10f;
3639     using std::modff;
3640     using std::powf;
3641     using std::sinf;
3642     using std::sinhf;
3643     using std::sqrtf;
3644     using std::tanf;
3645     using std::tanhf;

3646     using std::acos;
3648     using std::asin;
3649     using std::atan;
3650     using std::atan2;
3651     using std::ceil;
3652     using std::cos;
3653     using std::cosh;
```

```
3654     using std::exp;
3655     using std::fabs;
3656     using std::floor;
3657     using std::fmod;
3658     using std::frexp;
3659     using std::hypot;
3660     using std::ldexp;
3661     using std::log;
3662     using std::log10;
3663     using std::modf;
3664     using std::pow;
3665     using std::sin;
3666     using std::sinh;
3667     using std::sqrt;
3668     using std::tan;
3669     using std::tanh;
3670
3671     using std::acosh;
3672     using std::acoshf;
3673     using std::asinh;
3674     using std::asinhf;
3675     using std::atanh;
3676     using std::atanhf;
3677     using std::cbrt;
3678     using std::cbrtf;
3679     using std::copysign;
3680     using std::copysignf;
3681     using std::erf;
3682     using std::erfc;
3683     using std::erfcf;
3684     using std::erff;
3685     using std::exp2;
3686     using std::exp2f;
3687     using std::expm1;
3688     using std::expm1f;
3689     using std::fdim;
3690     using std::fdimf;
3691     using std::fma;
3692     using std::fmaf;
3693     using std::fmax;
3694     using std::fmaxf;
3695     using std::fmin;
3696     using std::fminf;
3697     using std::ilogb;
3698     using std::ilogbf;
3699     using std::log1p;
3700     using std::log1pf;
3701     using std::log2;
3702     using std::log2f;
3703     using std::logb;
3704     using std::logbf;
3705     using std::nearbyint;
3706     using std::nearbyintf;
3707     using std::nextafter;
3708     using std::nextafterf;
3709     using std::remainder;
3710     using std::remainderf;
3711     using std::remquo;
```

```

3712     using std::remquo;
3713     using std::round;
3714     using std::roundf;
3715     using std::scalbn;
3716     using std::scalbnf;
3717     using std::tgamma;
3718     using std::tgammaf;
3719     using std::trunc;
3720     using std::truncf;
3721

```

3722 Importing these names into the `precise_math` namespace enables each of them to be called in unqualified syntax from a  
 3723 function that has both “`restrict(cpu,amp)`” restrictions. E.g.,

```

3724
3725 void compute() restrict(cpu,amp) {
3726     ...
3727     float x = cos(y); // resolves to std::cos in “cpu” context; else fast_math::cos in “amp” context
3728     ...
3729 }
3730

```

### 3731 9.3 Miscellaneous Math Functions (Optional)

3732 The following functions allow access to Direct3D intrinsic functions. These are included in `<amp.h>` in the  
 3733 `concurrency::direct3d` namespace, and are only callable from a `restrict(amp)` function.

3734

|                                              |
|----------------------------------------------|
| <code>int abs(int val) restrict(amp);</code> |
|----------------------------------------------|

Returns the absolute value of the integer argument.

**Parameters:**

|                  |                  |
|------------------|------------------|
| <code>val</code> | The input value. |
|------------------|------------------|

Returns the absolute value of the input argument.

3735

|                                                                        |
|------------------------------------------------------------------------|
| <code>int clamp(int x, int min, int max) restrict(amp);</code>         |
| <code>float clamp(float x, float min, float max) restrict(amp);</code> |

Clamps the input argument “`x`” so it is always within the range  $[min,max]$ . If  $x < min$ , then this function returns the value of `min`. If  $x > max$ , then this function returns the value of `max`. Otherwise, `x` is returned.

**Parameters:**

|                  |                  |
|------------------|------------------|
| <code>val</code> | The input value. |
|------------------|------------------|

|                  |                                |
|------------------|--------------------------------|
| <code>min</code> | The minimum value of the range |
|------------------|--------------------------------|

|                  |                                |
|------------------|--------------------------------|
| <code>max</code> | The maximum value of the range |
|------------------|--------------------------------|

Returns the clamped value of “`x`”.

3736

|                                                                      |
|----------------------------------------------------------------------|
| <code>unsigned int countbits(unsigned int val) restrict(amp);</code> |
|----------------------------------------------------------------------|

Counts the number of bits in the input argument that are set (1).

**Parameters:**

|                  |                  |
|------------------|------------------|
| <code>val</code> | The input value. |
|------------------|------------------|

Returns the number of bits that are set.

3737

|                                                       |
|-------------------------------------------------------|
| <code>int firstbithigh(int val) restrict(amp);</code> |
|-------------------------------------------------------|

|                                                             |                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                             | Returns the bit position of the first set (1) bit in the input "val", starting from highest-order and working down.                                                                                                                                                                                                             |
| <b>Parameters:</b>                                          |                                                                                                                                                                                                                                                                                                                                 |
| val                                                         | The input value.                                                                                                                                                                                                                                                                                                                |
| Returns the position of the highest-order set bit in "val". |                                                                                                                                                                                                                                                                                                                                 |
| 3738                                                        | <pre>int firstbitlow(int val) restrict(amp);</pre> <p>Returns the bit position of the first set (1) bit in the input "val", starting from lowest-order and working up.</p>                                                                                                                                                      |
| <b>Parameters:</b>                                          |                                                                                                                                                                                                                                                                                                                                 |
| val                                                         | The input value.                                                                                                                                                                                                                                                                                                                |
| Returns the position of the lowest-order set bit in "val".  |                                                                                                                                                                                                                                                                                                                                 |
| 3739                                                        | <pre>int imax(int x, int y) restrict(amp); unsigned int umax(unsigned int x, unsigned int y) restrict(amp);</pre> <p>Returns the maximum of "x" and "y".</p>                                                                                                                                                                    |
| <b>Parameters:</b>                                          |                                                                                                                                                                                                                                                                                                                                 |
| X                                                           | The first input value.                                                                                                                                                                                                                                                                                                          |
| Y                                                           | The second input value                                                                                                                                                                                                                                                                                                          |
| Returns the maximum of the inputs.                          |                                                                                                                                                                                                                                                                                                                                 |
| 3740                                                        | <pre>int imin(int x, int y) restrict(amp); unsigned int umin(unsigned int x, unsigned int y) restrict(amp);</pre> <p>Returns the minimum of "x" and "y".</p>                                                                                                                                                                    |
| <b>Parameters:</b>                                          |                                                                                                                                                                                                                                                                                                                                 |
| x                                                           | The first input value.                                                                                                                                                                                                                                                                                                          |
| y                                                           | The second input value                                                                                                                                                                                                                                                                                                          |
| Returns the minimum of the inputs.                          |                                                                                                                                                                                                                                                                                                                                 |
| 3741                                                        | <pre>float mad(float x, float y, float z) restrict(amp); double mad(double x, double y, double z) restrict(amp); int mad(int x, int y, int z) restrict(amp); unsigned int mad(unsigned int x, unsigned int y, unsigned int z) restrict(amp);</pre> <p>Performs a multiply-add on the three arguments: <math>x*y + z</math>.</p> |
| <b>Parameters:</b>                                          |                                                                                                                                                                                                                                                                                                                                 |
| x                                                           | The first input multiplicand.                                                                                                                                                                                                                                                                                                   |
| y                                                           | The second input multiplicand                                                                                                                                                                                                                                                                                                   |
| z                                                           | The third input addend                                                                                                                                                                                                                                                                                                          |
| Returns $x*y + z$ .                                         |                                                                                                                                                                                                                                                                                                                                 |
| 3742                                                        | <pre>float noise(float x) restrict(amp);</pre>                                                                                                                                                                                                                                                                                  |

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |            |                                 |            |                                 |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|---------------------------------|------------|---------------------------------|
|            | <p>Generates a random value using the Perlin noise algorithm. The returned value will be within the range [-1,+1].</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>x</i></td><td>The first input value.</td></tr> </table> <p>Returns the random noise value.</p>                                                                                                                                                                        | <i>x</i>   | The first input value.          |            |                                 |
| <i>x</i>   | The first input value.                                                                                                                                                                                                                                                                                                                                                                                                                                |            |                                 |            |                                 |
| 3743       | <pre>float radians(float x) restrict(amp);</pre> <p>Converts from "x" degrees into radians.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>x</i></td><td>The input value in degrees.</td></tr> </table> <p>Returns the radian value.</p>                                                                                                                                                                                                | <i>x</i>   | The input value in degrees.     |            |                                 |
| <i>x</i>   | The input value in degrees.                                                                                                                                                                                                                                                                                                                                                                                                                           |            |                                 |            |                                 |
| 3744       | <pre>float rcp(float x) restrict(amp);</pre> <p>Calculates a fast approximate reciprocal of "x".</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>x</i></td><td>The input value.</td></tr> </table> <p>Returns the reciprocal of the input.</p>                                                                                                                                                                                           | <i>x</i>   | The input value.                |            |                                 |
| <i>x</i>   | The input value.                                                                                                                                                                                                                                                                                                                                                                                                                                      |            |                                 |            |                                 |
| 3745       | <pre>unsigned int reversebits(unsigned int val) restrict(amp);</pre> <p>Reverses the order of the bits in the input argument.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>val</i></td><td>The input value.</td></tr> </table> <p>Returns the bit-reversed number.</p>                                                                                                                                                                | <i>val</i> | The input value.                |            |                                 |
| <i>val</i> | The input value.                                                                                                                                                                                                                                                                                                                                                                                                                                      |            |                                 |            |                                 |
| 3746       | <pre>float saturate(float x) restrict(amp);</pre> <p>Clamps the input value into the range [0,1].</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>x</i></td><td>The input value.</td></tr> </table> <p>Returns the clamped value.</p>                                                                                                                                                                                                    | <i>x</i>   | The input value.                |            |                                 |
| <i>x</i>   | The input value.                                                                                                                                                                                                                                                                                                                                                                                                                                      |            |                                 |            |                                 |
| 3747       | <pre>int sign(int x) restrict(amp);</pre> <p>Returns the sign of "x"; that is, it returns -1 if <i>x</i> is negative, 0 if <i>x</i> is 0, or +1 if <i>x</i> is positive.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>x</i></td><td>The first input value.</td></tr> </table> <p>Returns the sign of the input.</p>                                                                                                                   | <i>x</i>   | The first input value.          |            |                                 |
| <i>x</i>   | The first input value.                                                                                                                                                                                                                                                                                                                                                                                                                                |            |                                 |            |                                 |
| 3748       | <pre>float smoothstep(float min, float max, float x) restrict(amp);</pre> <p>Returns a smooth Hermite interpolation between 0 and 1, if <i>x</i> is in the range [min, max]; 0 if <i>x</i> is less than min; 1 if <i>x</i> is greater than max.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td><i>min</i></td><td>The minimum value of the range.</td></tr> <tr> <td><i>max</i></td><td>The maximum value of the range.</td></tr> </table> | <i>min</i> | The minimum value of the range. | <i>max</i> | The maximum value of the range. |
| <i>min</i> | The minimum value of the range.                                                                                                                                                                                                                                                                                                                                                                                                                       |            |                                 |            |                                 |
| <i>max</i> | The maximum value of the range.                                                                                                                                                                                                                                                                                                                                                                                                                       |            |                                 |            |                                 |

|                                 |                               |
|---------------------------------|-------------------------------|
| <code>x</code>                  | The value to be interpolated. |
| Returns the interpolated value. |                               |

3749

`float step(float y, float x) restrict(amp);`

Compares two values, returning 0 or 1 based on which value is greater.

**Parameters:**

|                                                                                                                   |                         |
|-------------------------------------------------------------------------------------------------------------------|-------------------------|
| <code>y</code>                                                                                                    | The first input value.  |
| <code>x</code>                                                                                                    | The second input value. |
| Returns 1 if the <code>x</code> parameter is greater than or equal to the <code>y</code> parameter; otherwise, 0. |                         |

3750

`uint4 msad4(uint reference, uint2 source, uint4 accum) restrict(amp);`

Compares a 4-byte reference value and an 8-byte source value and accumulates a vector of 4 sums. Each sum corresponds to the masked sum of absolute differences of different byte alignments between the reference value and the source value.

**Parameters:**

|                                                                                                                                                                           |                                                                                                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>reference</code>                                                                                                                                                    | The reference array of 4 bytes in one uint value                                                                                                              |
| <code>source</code>                                                                                                                                                       | The source array of 8 bytes in a vector of two uint values.                                                                                                   |
| <code>accum</code>                                                                                                                                                        | A vector of 4 values to be added to the masked sum of absolute differences of the different byte alignments between the reference value and the source value. |
| Returns a vector of 4 sums. Each sum corresponds to the masked sum of absolute differences of different byte alignments between the reference value and the source value. |                                                                                                                                                               |

3751

## 3752 10 Graphics (Optional)

3753 Programming model elements defined in `<amp_graphics.h>` and `<amp_short_vectors.h>` are designed for graphics

3754 programming in conjunction with accelerated compute on an accelerator, and are therefore appropriate only for proper

3755 GPU accelerators. Accelerators that do not support native graphics functionality need not implement these features.

3756

3757 All types in this section are defined in the `concurrency::graphics` namespace.

### 3758 10.1 texture<T,N>

3759 The `texture` class provides the means to create textures from raw memory or from file. `textures` are similar to `arrays` in that

3760 they are containers of data and they behave like STL containers with respect to assignment and copy construction.

3761

3762 `textures` are templated on `T`, the element type, and on `N`, the rank of the texture. `N` can be one of 1, 2 or 3.

3763

3764 The element type of the `texture`, also referred to as the texture's logical element type, is one of a closed set of short vector3765 types defined in the `concurrency::graphics` namespace and covered elsewhere in this specification. The below table briefly

3766 enumerates all supported element types.

3767

| Rank of element type, (also referred to as "number of scalar elements") | Signed Integer | Unsigned Integer | Single precision floating point number | Single precision signed normalized number | Single precision unsigned normalized number | Double precision floating point number |
|-------------------------------------------------------------------------|----------------|------------------|----------------------------------------|-------------------------------------------|---------------------------------------------|----------------------------------------|
| 1                                                                       | int            | unsigned int     | float                                  | norm                                      | unorm                                       | double                                 |
| 2                                                                       | int_2          | uint_2           | float_2                                | norm_2                                    | unorm_2                                     | double_2                               |

|   |       |        |         |        |         |          |
|---|-------|--------|---------|--------|---------|----------|
| 3 | int_3 | uint_3 | float_3 | norm_3 | unorm_3 | double_3 |
| 4 | int_4 | uint_4 | float_4 | norm_4 | unorm_4 | double_4 |

3768

3769

3770 Remarks:

- 3771 1. *norm* and *unorm* vector types are vector of *floats* which are normalized to the range [-1..1] and [0...1], respectively.
- 3772 2. Grayed-out cells represent vector types which are defined by C++ AMP but which are not supported as *texture* value types. Implementations can optionally support the types in the grayed-out cells in the above table.

3774 **Microsoft-specific:** grayed-out cells in the above table are not supported.3775 **10.1.1 Synopsis**

```

3776
3777 template <typename T, int N>
3778 class texture
3779 {
3780 public:
3781     static const int rank = N;
3782     typedef typename T value_type;
3783     typedef short_vectors_traits<T>::scalar_type scalar_type;
3784
3785     texture(const extent<N>& ext);
3786
3787     texture(int e0);
3788     texture(int e0, int e1);
3789     texture(int e0, int e1, int e2);
3790
3791     texture(const extent<N>& ext, const accelerator_view& acc_view);
3792     texture(const extent<N>& ext, const accelerator_view& av, const accelerator_view&
3793 associated_av);
3794
3795     texture(int e0, const accelerator_view& acc_view);
3796     texture(int e0, const accelerator_view& av, const accelerator_view& associated_av);
3797     texture(int e0, int e1, const accelerator_view& acc_view);
3798     texture(int e0, int e1, const accelerator_view& av, const accelerator_view& associated_av);
3799     texture(int e0, int e1, int e2, const accelerator_view& acc_view);
3800     texture(int e0, int e1, int e2, const accelerator_view& av, const accelerator_view&
3801 associated_av);
3802
3803     texture(const extent<N>& ext, unsigned int bits_per_scalar_element, unsigned int
3804 mip_levels);
3805
3806     texture(const extent<N>& ext, unsigned int bits_per_scalar_element);
3807
3808     texture(int e0, unsigned int bits_per_scalar_element);
3809     texture(int e0, int e1, unsigned int bits_per_scalar_element);
3810     texture(int e0, int e1, int e2, unsigned int bits_per_scalar_element);
3811
3812     texture(const extent<N>& ext, unsigned int bits_per_scalar_element, unsigned int mip_levels,
3813 const accelerator_view& acc_view);
3814
3815     texture(const extent<N>& ext, unsigned int bits_per_scalar_element, const accelerator_view&
3816 acc_view);
3817
3818     texture(int e0, unsigned int bits_per_scalar_element, const accelerator_view& acc_view);

```

```

3819    texture(int e0, int e1, unsigned int bits_per_scalar_element,
3820            const accelerator_view& acc_view);
3821
3822    texture(int e0, int e1, int e2, unsigned int bits_per_scalar_element,
3823            const accelerator_view& acc_view);
3824
3825    texture(const extent<N>& ext, unsigned int bits_per_scalar_element,
3826            const accelerator_view& av, const accelerator_view& associated_av);
3827
3828    texture(int e0, unsigned int bits_per_scalar_element, const accelerator_view& av,
3829            const accelerator_view& associated_av);
3829
3830    texture(int e0, int e1, unsigned int bits_per_scalar_element,
3831            const accelerator_view& av, const accelerator_view& associated_av);
3831
3832    texture(int e0, int e1, int e2, unsigned int bits_per_scalar_element,
3833            const accelerator_view& av, const accelerator_view& associated_av);
3834
3835    template <typename TInputIterator>
3836        texture(const extent<N>& ext, TInputIterator src_first, TInputIterator src_last);
3837
3838    template <typename TInputIterator>
3839        texture(int e0, TInputIterator src_first, TInputIterator src_last);
3840
3841    template <typename TInputIterator>
3842        texture(int e0, int e1, TInputIterator src_first, TInputIterator src_last);
3843
3844    template <typename TInputIterator>
3845        texture(int e0, int e1, int e2, TInputIterator src_first,
3846                TInputIterator src_last);
3847
3848    template <typename TInputIterator>
3849        texture(const extent<N>& ext, TInputIterator src_first, TInputIterator src_last,
3850                const accelerator_view& acc_view);
3851
3852    template <typename TInputIterator>
3853        texture(int e0, TInputIterator src_first, TInputIterator src_last,
3854                const accelerator_view& acc_view);
3855
3856    template <typename TInputIterator>
3857        texture(int e0, int e1, TInputIterator src_first, TInputIterator src_last,
3858                const accelerator_view& acc_view);
3859
3860    texture(int e0, int e1, int e2, TInputIterator src_first, TInputIterator src_last, const
3861           accelerator_view& acc_view);
3862
3863    template <typename TInputIterator>
3864        texture(const extent<N>& ext, TInputIterator src_first, TInputIterator src_last,
3865                const accelerator_view& av, const accelerator_view& associated_av);
3866
3867    template <typename TInputIterator>
3868        texture(int e0, TInputIterator src_first, TInputIterator src_last,
3869                const accelerator_view& av, const accelerator_view& associated_av);
3870
3871    template <typename TInputIterator>
3872        texture(int e0, int e1, TInputIterator src_first, TInputIterator src_last,
3873                const accelerator_view& av, const accelerator_view& associated_av);
3874
3875    template <typename TInputIterator>
3876        texture(int e0, int e1, TInputIterator src_first, TInputIterator src_last,
3877                const accelerator_view& av, const accelerator_view& associated_av);
3878

```

```

3877     texture(int e0, int e1, int e2, TInputIterator src_first, TInputIterator src_last,
3878             const accelerator_view& av, const accelerator_view& associated_av);
3879
3880     texture(const extent<N>& ext, const void * source, unsigned int src_byte_size,
3881             unsigned int bits_per_scalar_element);
3882
3883     texture(int e0, const void * source, unsigned int src_byte_size,
3884             unsigned int bits_per_scalar_element);
3885
3886     texture(int e0, int e1, const void * source, unsigned int src_byte_size,
3887             unsigned int bits_per_scalar_element);
3888
3889     texture(int e0, int e1, int e2, const void * source,
3890             unsigned int src_byte_size, unsigned int bits_per_scalar_element);
3891
3892     texture(const extent<N>& ext, const void * source, unsigned int src_byte_size,
3893             unsigned int bits_per_scalar_element, const accelerator_view& acc_view);
3894
3895     texture(int e0, const void * source, unsigned int src_byte_size,
3896             unsigned int bits_per_scalar_element, const accelerator_view& acc_view);
3897
3898     texture(int e0, int e1, const void * source, unsigned int src_byte_size,
3899             unsigned int bits_per_scalar_element, const accelerator_view& acc_view);
3900
3901     texture(int e0, int e1, int e2, const void * source, unsigned int src_byte_size,
3902             unsigned int bits_per_scalar_element, const accelerator_view& acc_view);
3903
3904     texture(const extent<N>& ext, const void * source, unsigned int src_byte_size,
3905             unsigned int bits_per_scalar_element, const accelerator_view& av, const
3906             accelerator_view& associated_av);
3907
3908     texture(int e0, const void * source, unsigned int src_byte_size,
3909             unsigned int bits_per_scalar_element, const accelerator_view& av, const
3910             accelerator_view& associated_av);
3911
3912     texture(int e0, int e1, const void * source, unsigned int src_byte_size,
3913             unsigned int bits_per_scalar_element, const accelerator_view& av, const
3914             accelerator_view& associated_av);
3915
3916     texture(int e0, int e1, int e2, const void * source, unsigned int src_byte_size,
3917             unsigned int bits_per_scalar_element, const accelerator_view& av, const
3918             accelerator_view& associated_av);
3919
3920     texture(const texture& src);
3921     texture(const texture& src, const accelerator_view& acc_view);
3922     texture(const texture& src, const accelerator_view& av, const accelerator_view&
3923             associated_av);
3924
3925     texture(const texture_view<value_type, rank> & src);
3926     texture(const texture_view<value_type, rank> & src, const Concurrency::accelerator_view &
3927             acc_view);
3928
3929     texture(const texture_view<value_type, rank> & src, const accelerator_view& av, const
3930             accelerator_view& associated_av);
3931
3932     texture(const texture_view<const value_type, rank> & src);
3933     texture(const texture_view<const value_type, rank> & src, const
3934             Concurrency::accelerator_view & acc_view);

```

```

3935     texture(const texture_view<const value_type, rank> & src, const accelerator_view& av, const
3936     accelerator_view& associated_av);
3937
3938     texture& operator=(const texture& src);
3939
3940     texture(texture&& other);
3941     texture& operator=(texture&& other);
3942
3943     void copy_to(texture& dest) const;
3944     void copy_to(const writeonly_texture_view<T,N>& dest) const;
3945
3946     void* data();
3947     const void* data() const;
3948
3949 // Microsoft-specific:
3950     __declspec(property(get=get_row_pitch)) unsigned int row_pitch;
3951     __declspec(property(get=get_depth_pitch)) unsigned int depth_pitch;
3952     __declspec(property(get=get_bits_per_scalar_element)) unsigned int bits_per_scalar_element;
3953     __declspec(property(get=get_mipmap_levels)) unsigned int mipmap_levels;
3954     __declspec(property(get=get_data_length)) unsigned int data_length;
3955     __declspec(property(get=get_extent)) extent<N> extent;
3956     __declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
3957     __declspec(property(get=get_associated_accelerator_view)) accelerator_view
3958     associated_accelerator_view;
3959
3960     unsigned int get_row_pitch() const;
3961     unsigned int get_depth_pitch() const;
3962     unsigned int get_bits_per_scalar_element() const;
3963     unsigned int get_mipmap_levels() const restrict(cpu,amp);
3964     unsigned int get_data_length() const;
3965     extent<N> get_extent() const restrict(cpu,amp);
3966     extent<N> get_mipmap_extent(unsigned int mipmap_level) const restrict(cpu,amp);
3967     accelerator_view get_accelerator_view() const;
3968     accelerator_view get_associated_accelerator_view() const;
3969
3970     const value_type operator[] (const index<N>& index) const restrict(amp);
3971     const value_type operator[] (int i0) const restrict(amp);
3972     const value_type operator() (const index<N>& index) const restrict(amp);
3973     const value_type operator() (int i0) const restrict(amp);
3974     const value_type operator() (int i0, int i1) const restrict(amp);
3975     const value_type operator() (int i0, int i1, int i2) const restrict(amp);
3976     const value_type get(const index<N>& index) const restrict(amp);
3977
3978     void set(const index<N>& index, const value_type& val) restrict(amp);
3979 }

```

### 3980 10.1.2 Introduced typedefs

|                                |
|--------------------------------|
| <b>typedef ... value_type;</b> |
|--------------------------------|

The logical value type of the texture. e.g., for texture <float2, 3>, value\_type would be float2.

3981

|                                 |
|---------------------------------|
| <b>typedef ... scalar_type;</b> |
|---------------------------------|

The scalar type that serves as the component of the texture's value type. For example, for texture<int2, 3>, the scalar type would be "int".

3982  
3983

### 10.1.3 Constructing an uninitialized texture

```

texture(const extent<N>& ext);

texture(int e0);
texture(int e0, int e1);
texture(int e0, int e1, int e2);

texture(const extent<N>& ext, const accelerator_view& acc_view);

texture(int e0, const accelerator_view& acc_view);
texture(int e0, int e1, const accelerator_view& acc_view);
texture(int e0, int e1, int e2, const accelerator_view& acc_view);

texture(const extent<N>& ext, unsigned int bits_per_scalar_element);
texture(const extent<N>& ext, unsigned int bits_per_scalar_element, unsigned int mip_levels);

texture(int e0, unsigned int bits_per_scalar_element);
texture(int e0, int e1, unsigned int bits_per_scalar_element);
texture(int e0, int e1, int e2, unsigned int bits_per_scalar_element);

texture(const extent<N>& ext, unsigned int bits_per_scalar_element, const accelerator_view&
acc_view);

texture(const extent<N>& ext, unsigned int bits_per_scalar_element, unsigned int mip_levels,
const accelerator_view& acc_view);

texture(int e0, unsigned int bits_per_scalar_element, const accelerator_view& acc_view);
texture(int e0, int e1, unsigned int bits_per_scalar_element, const accelerator_view& acc_view);

texture(int e0, int e1, int e2, unsigned int bits_per_scalar_element, const accelerator_view&
acc_view);

```

Creates an uninitialized texture with the specified shape, number of bits per scalar element, on the specified accelerator view.

**Parameters:**

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ext                               | Extents of the texture to create                                                                                                                                                                                                                                                                                                                                                                                               |
| e0                                | Extent of dimension 0                                                                                                                                                                                                                                                                                                                                                                                                          |
| e1                                | Extent of dimension 1                                                                                                                                                                                                                                                                                                                                                                                                          |
| e2                                | Extent of dimension 2                                                                                                                                                                                                                                                                                                                                                                                                          |
| bits_per_scalar_element           | Number of bits per each scalar element in the underlying scalar type of the texture.                                                                                                                                                                                                                                                                                                                                           |
| mip_levels                        | Number of mipmaps in the texture. <ul style="list-style-type: none"> <li>The default value is 1 for all other constructors that do not specify mip_levels, meaning that the texture would hold a single texture image with original size;</li> <li>Value 0, will cause constructor to generate the full set of uninitialized mipmaps.</li> <li>Value greater than 0, will generate the specified number of mipmaps.</li> </ul> |
| acc_view                          | Accelerator view where to create the texture                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Error condition</b>            | <b>Exception thrown</b>                                                                                                                                                                                                                                                                                                                                                                                                        |
| Out of memory                     | concurrency::runtime_exception                                                                                                                                                                                                                                                                                                                                                                                                 |
| Invalid number of bits per scalar | concurrency::runtime_exception                                                                                                                                                                                                                                                                                                                                                                                                 |

|                                                               |                                  |
|---------------------------------------------------------------|----------------------------------|
| elementspecified                                              |                                  |
| Invalid number of mip levels specified.                       | concurrency::runtime_exception   |
| Invalid combination of value_type and bits per scalar element | concurrency::unsupported_feature |
| accelerator_view doesn't support textures                     | concurrency::unsupported_feature |

3984

3985 The table below summarizes all valid combinations of underlying scalar types (columns), ranks(rows), supported values for  
 3986 bits-per-scalar-element (inside the table cells), and default value of bits-per-scalar-element for each given combination  
 3987 (highlighted in green). Note that unorm and norm have no default value for bits-per-scalar-element. Implementations can  
 3988 optionally support textures of norm3 or unorm3 with no default bits-per-scalar-element value, or double3 or double4, with  
 3989 implementation-specific values of bits-per-scalar-element.

3990

3991 ***Microsoft-specific:*** the current implementation doesn't support textures of norm3, unorm3, double3, or double4.

3992

| Rank | int       | uint      | float  | norm  | unorm | double |
|------|-----------|-----------|--------|-------|-------|--------|
| 1    | 8, 16, 32 | 8, 16, 32 | 16, 32 | 8, 16 | 8, 16 | 64     |
| 2    | 8, 16, 32 | 8, 16, 32 | 16, 32 | 8, 16 | 8, 16 | 64     |
| 3    | 32        | 32        | 32     |       |       |        |
| 4    | 8, 16, 32 | 8, 16, 32 | 16, 32 | 8, 16 | 8, 16 |        |

3993

#### 3994 10.1.4 Constructing a staging texture

3995

3996 Staging textures are used as a hint to optimize repeated copies between two accelerators. Staging textures are optimized  
 3997 for data transfers, and do not have stable user-space memory.

3998

3999 ***Microsoft-specific:*** On Windows, staging textures are backed by DirectX staging textures which have the correct hardware  
 4000 alignment to ensure efficient DMA transfer between the CPU and a device.

4001

4002 Staging textures are differentiated from normal textures by their construction with a second accelerator\_view. Note that  
 4003 the [accelerator\\_view](#) property of a staging texture returns the value of the first accelerator\_view argument it was  
 4004 constructed with ([av](#), below).

4005

4006 It is illegal to change or examine the contents of a staging texture while it is involved in a transfer operation (i.e., between  
 4007 lines 17 and 22 in the following example):

4008

```

1. class SimulationServer
2. {
3.     texture<float,2> acceleratorTexture;
4.     texture<float,2> stagingTexture;
5. public:
6.     SimulationServer(const accelerator_view& av)
7.         :acceleratorTexture(extent<2>(1000,1000), av),
8.          stagingTexture(extent<2>(1000,1000), accelerator("cpu").default_view,
9.                         accelerator("gpu").default_view)
10.    {
11.    }
12.
13.    void OnCompute()
14.    {
15.        texture<float,2> &t = acceleratorTexture;
16.        LoadData(stagingTexture.data(), stagingTexture.row_pitch);
17.        completion_future cf = copy_async(stagingTexture, t);

```

```

4026
4027     18. // Illegal to access stagingTexture here
4028
4029     19. cf.wait();
4030     20. parallel_for_each(t.extent, [&](index<2> idx)
4031     {
4032         21.     // Update texture "t" according to simulation
4033         22.     }
4034         23.     completion_future cf1 = copy_async(t, stagingTexture);
4035
4036         24.     // Illegal to access stagingTexture here
4037         25.     }
4038         26.     cf1.wait();
4039         27.     SendToClient(stagingTexture.data(), stagingTexture.row_pitch);
4040     28.     }
4041     29.     }
4042     30.     }
4043     31.     }
4044     32.     }
4045     33. };

```

**texture(const extent<N>& ext, const accelerator\_view& av, const accelerator\_view& associated\_av);**

**texture(int e0, const accelerator\_view& av, const accelerator\_view& associated\_av);**

**texture(int e0, int e1, const accelerator\_view& av, const accelerator\_view& associated\_av);**

**texture(int e0, int e1, int e2, const accelerator\_view& ev, const accelerator\_view& associated\_av);**

**texture(const extent<N>& ext, unsigned int bits\_per\_scalar\_element, const accelerator\_view& av, const accelerator\_view& associated\_av);**

**texture(int e0, unsigned int bits\_per\_scalar\_element, const accelerator\_view& av, const accelerator\_view& associated\_av);**

**texture(int e0, int e1, unsigned int bits\_per\_scalar\_element, const accelerator\_view& av, const accelerator\_view& associated\_av);**

**texture(int e0, int e1, int e2, unsigned int bits\_per\_scalar\_element, const accelerator\_view& av, const accelerator\_view& associated\_av);**

Constructs a staging texture with the given extent, which acts as a staging area between accelerator views "av" and "associated\_av". If "av" is a cpu accelerator view, this will construct a staging texture which is optimized for data transfers between the CPU and "associated\_av".

#### Parameters:

|                                                      |                                                                                                                                  |
|------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| ext                                                  | Extents of the texture to create                                                                                                 |
| e0                                                   | Extent of dimension 0                                                                                                            |
| e1                                                   | Extent of dimension 1                                                                                                            |
| e2                                                   | Extent of dimension 2                                                                                                            |
| bits_per_scalar_element                              | Number of bits per each scalar element in the underlying scalar type of the texture.                                             |
| av                                                   | An accelerator_view object which specifies the home location of this texture.                                                    |
| associated_av                                        | An accelerator_view object which specifies a target accelerator_view that this staging texture is optimized for copying to/from. |
| <b>Error condition</b>                               | <b>Exception thrown</b>                                                                                                          |
| Out of memory                                        | concurrency::runtime_exception                                                                                                   |
| Invalid number of bits per scalar elements specified | concurrency::runtime_exception                                                                                                   |
| Invalid combination of                               | concurrency::unsupported_feature                                                                                                 |

|                                           |                                  |
|-------------------------------------------|----------------------------------|
| value_type and bits per scalar element    |                                  |
| accelerator_view doesn't support textures | concurrency::unsupported_feature |

4044

### 10.1.5 Constructing a texture from a host side iterator

4045

```

template <typename TInputIterator>
texture(const extent<N>& ext, TInputIterator src_first, TInputIterator src_last);
texture(int e0, TInputIterator src_first, TInputIterator src_last);
texture(int e0, int e1, TInputIterator src_first, TInputIterator src_last);
texture(int e0, int e1, int e2, TInputIterator src_first, TInputIterator src_last);
template <typename TInputIterator>
texture(const extent<N>& ext, TInputIterator src_first, TInputIterator src_last, const
accelerator_view& acc_view);

texture(int e0, TInputIterator src_first, TInputIterator src_last, const accelerator_view&
acc_view);

texture(int e0, int e1, TInputIterator src_first, TInputIterator src_last, const
accelerator_view& acc_view);

texture(int e0, int e1, int e2, TInputIterator src_first, TInputIterator src_last, const
accelerator_view& acc_view);

template <typename TInputIterator>
texture(const extent<N>& ext, TInputIterator src_first, TInputIterator src_last, const
accelerator_view& av, const accelerator_view& associated_av);

texture(int e0, TInputIterator src_first, TInputIterator src_last, const accelerator_view& av,
const accelerator_view& associated_av);

texture(int e0, int e1, TInputIterator src_first, TInputIterator src_last, const
accelerator_view& av, const accelerator_view& associated_av);

texture(int e0, int e1, int e2, TInputIterator src_first, TInputIterator src_last, const
accelerator_view& av, const accelerator_view& associated_av);

```

Creates a texture from a host-side iterator. The data type of the iterator must be the same as the value type of the texture. Textures with element types based on norm or unorm do not support this constructor (usage of it will result in a compile-time error).

**Parameters:**

|                        |                                                                                                                                  |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| ext                    | Extents of the texture to create                                                                                                 |
| e0                     | Extent of dimension 0                                                                                                            |
| e1                     | Extent of dimension 1                                                                                                            |
| e2                     | Extent of dimension 2                                                                                                            |
| src_first              | Iterator pointing to the first element to be copied into the texture                                                             |
| src_last               | Iterator pointing immediately past the last element to be copied into the texture                                                |
| av                     | An accelerator_view object which specifies the home location of this texture.                                                    |
| associated_av          | An accelerator_view object which specifies a target accelerator_view that this staging texture is optimized for copying to/from. |
| <b>Error condition</b> | <b>Exception thrown</b>                                                                                                          |
| Out of memory          | concurrency::runtime_exception                                                                                                   |
| Inadequate amount      | concurrency::runtime_exception                                                                                                   |

|                                           |                                  |
|-------------------------------------------|----------------------------------|
| of data supplied through the iterators    |                                  |
| Accelerator_view doesn't support textures | concurrency::unsupported_feature |

4046

#### 10.1.6 Constructing a texture from a host-side data source

4048

```

texture(const extent<N>& ext, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element);

texture(int e0, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element);

texture(int e0, int e1, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element);

texture(int e0, int e1, int e2, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element);

texture(const extent<N>& ext, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element, const accelerator_view& acc_view);

texture(int e0, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element, const accelerator_view& acc_view);

texture(int e0, int e1, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element, const accelerator_view& acc_view);

texture(int e0, int e1, int e2, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element, const accelerator_view& acc_view);

texture(const extent<N>& ext, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element, const accelerator_view& av, const accelerator_view& associated_av);

texture(int e0, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element, const accelerator_view& av, const accelerator_view& associated_av);

texture(int e0, int e1, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element, const accelerator_view& av, const accelerator_view& associated_av);

texture(int e0, int e1, int e2, const void * source, unsigned int src_byte_size, unsigned int
bits_per_scalar_element, const accelerator_view& av, const accelerator_view& associated_av);

```

Creates a texture from a host-side provided buffer. The format of the data source must be compatible with the texture's scalar type, and the amount of data in the data source must be exactly the amount necessary to initialize a texture in the specified format, with the given number of bits per scalar element.

For example, a 2D texture of uint2 initialized with the extent of 100x200 and with bits\_per\_scalar\_element equal to 8 will require a total of  $100 * 200 * 2 * 8 = 320,000$  bits available to copy from source, which is equal to 40,000 bytes. (or in other words, one byte, per one scalar element, for each scalar element, and each pixel, in the texture).

**Parameters:**

|     |                                  |
|-----|----------------------------------|
| ext | Extents of the texture to create |
|-----|----------------------------------|

|                                                                                                  |                                                                                                                                  |
|--------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| e0                                                                                               | Extent of dimension 0                                                                                                            |
| e1                                                                                               | Extent of dimension 1                                                                                                            |
| e2                                                                                               | Extent of dimension 2                                                                                                            |
| source                                                                                           | Pointer to a host buffer                                                                                                         |
| src_byte_size                                                                                    | Number of bytes of the host source buffer                                                                                        |
| bits_per_scalar_element                                                                          | Number of bits per each scalar element in the underlying scalar type of the texture.                                             |
| av                                                                                               | An accelerator_view object which specifies the home location of this texture.                                                    |
| associated_av                                                                                    | An accelerator_view object which specifies a target accelerator_view that this staging texture is optimized for copying to/from. |
| <b>Error condition</b>                                                                           | <b>Exception thrown</b>                                                                                                          |
| Out of memory                                                                                    | concurrency::runtime_exception                                                                                                   |
| Inadequate amount of data supplied through the host buffer (src_byte_size < texture.data_length) | concurrency::runtime_exception                                                                                                   |
| Invalid number of bits per scalar elementspecified                                               | concurrency::runtime_exception                                                                                                   |
| Invalid combination of value_type and bits per scalar element                                    | concurrency::unsupported_feature                                                                                                 |
| Accelerator_view doesn't support textures                                                        | concurrency::unsupported_feature                                                                                                 |

4049

4050

### 10.1.7 Constructing a texture by cloning another

4051

```
texture(const texture& src);
texture(const texture_view<value_type, rank>& src);
texture(const texture_view<const value_type, rank>& src);
```

Initializes one texture from another. The texture is created on the same accelerator view as the source.

**Parameters:**

|     |                                             |
|-----|---------------------------------------------|
| src | Source texture or texture_view to copy from |
|-----|---------------------------------------------|

|                        |                         |
|------------------------|-------------------------|
| <b>Error condition</b> | <b>Exception thrown</b> |
|------------------------|-------------------------|

|               |                                |
|---------------|--------------------------------|
| Out of memory | concurrency::runtime_exception |
|---------------|--------------------------------|

4052

```
texture(const texture& src, const accelerator_view& acc_view);
texture(const texture_view<value_type, rank>& src, const accelerator_view& acc_view);
texture(const texture_view<const value_type, rank>& src, const accelerator_view& acc_view);
```

Initializes one texture from another.

**Parameters:**

|     |                                             |
|-----|---------------------------------------------|
| src | Source texture or texture_view to copy from |
|-----|---------------------------------------------|

|          |                                              |
|----------|----------------------------------------------|
| acc_view | Accelerator view where to create the texture |
|----------|----------------------------------------------|

|                        |                         |
|------------------------|-------------------------|
| <b>Error condition</b> | <b>Exception thrown</b> |
|------------------------|-------------------------|

|               |                                |
|---------------|--------------------------------|
| Out of memory | concurrency::runtime_exception |
|---------------|--------------------------------|

|                                           |                                  |
|-------------------------------------------|----------------------------------|
| Accelerator_view doesn't support textures | concurrency::unsupported_feature |
|-------------------------------------------|----------------------------------|

4053

```
texture(const texture& src, const accelerator_view& av, const accelerator_view& associated_av);

texture(const texture_view<value_type, rank>& src, const accelerator_view& av, const
accelerator_view& associated_av);

texture(const texture_view<const value_type, rank>& src, const accelerator_view& av, const
accelerator_view& associated_av);
```

Initializes a staging texture from another. The source texture could be a staging texture as well.

**Parameters:**

|                                           |                                                                                                                                  |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| src                                       | Source texture or texture_view to copy from                                                                                      |
| av                                        | An accelerator_view object which specifies the home location of this texture.                                                    |
| associated_av                             | An accelerator_view object which specifies a target accelerator_view that this staging texture is optimized for copying to/from. |
| <b>Error condition</b>                    | <b>Exception thrown</b>                                                                                                          |
| Out of memory                             | concurrency::runtime_exception                                                                                                   |
| Accelerator_view doesn't support textures | concurrency::unsupported_feature                                                                                                 |

4054

#### 4055 10.1.8 Assignment operator

4056

```
texture& operator=(const texture& src);
```

Release the resource of this texture, allocate the resource according to src's properties, then deep copy src's content to this texture.

**Parameters:**

|                        |                                             |
|------------------------|---------------------------------------------|
| src                    | Source texture or texture_view to copy from |
| <b>Error condition</b> | <b>Exception thrown</b>                     |
| Out of memory          | concurrency::runtime_exception              |

4057

#### 4058 10.1.9 Copying textures

```
void copy_to(texture& dest) const;
void copy_to(const writeonly_texture_view<T,N>& dest) const;
```

Copies the contents of one texture onto the other. The textures must have been created with exactly the same extent and with compatible physical formats; that is, the number of mipmap levels, the number of scalar elements and the number of bits per scalar elements must agree. The textures could be from different accelerators. For copying to writeonly\_texture\_view the texture cannot have multiple mipmap levels.

**Parameters:**

|                                     |                                                          |
|-------------------------------------|----------------------------------------------------------|
| dest                                | Destination texture or writeonly_texture_view to copy to |
| <b>Error condition</b>              | <b>Exception thrown</b>                                  |
| Out of memory                       | concurrency::runtime_exception                           |
| Incompatible texture formats        | concurrency::runtime_exception                           |
| Extents don't match                 | concurrency::runtime_exception                           |
| Number of mipmap levels don't match | concurrency::runtime_exception                           |

4059

```
void* data();
const void* data() const;
```

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |              |                                                     |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|-----------------------------------------------------|
|              | Returns a pointer to the raw data underlying the staging texture. For non-staging texture it will return nullptr.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |              |                                                     |
|              | <b>Return Value:</b><br>A (const) pointer to the first byte of the raw data underlying the staging texture.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |              |                                                     |
| 4060         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |              |                                                     |
| 4061         | <b>10.1.10 Moving textures</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |              |                                                     |
| 4062         | <pre>texture(texture&amp;&amp; other); texture&amp; operator=(texture&amp;&amp; other);</pre> <p>"Moves" (in the C++ rvalue reference sense) the contents of other to "this". The source and destination textures do not have to be necessarily on the same accelerator originally.</p> <p>As is typical in C++ move constructors, no actual copying or data movement occurs; simply one C++ texture object is vacated of its internal representation, which is moved to the target C++ texture object.</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td>other</td> <td>Object whose contents are moved to "this"</td> </tr> </table> <p><b>Error condition: none</b></p> | other        | Object whose contents are moved to "this"           |
| other        | Object whose contents are moved to "this"                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |              |                                                     |
| 4063         | <b>10.1.11 Querying texture's physical characteristics</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |              |                                                     |
| 4064         | <pre>unsigned int get_bits_per_scalar_element() const; __declspec(property(get=get_bits_per_scalar_element)) unsigned int bits_per_scalar_element;</pre> <p>Gets the bits-per-scalar-element of the texture. Returns 0, if the texture is created using Direct3D Interop (10.1.16).</p> <p><b>Error conditions: none</b></p>                                                                                                                                                                                                                                                                                                                                                       |              |                                                     |
| 4065         | <pre>unsigned int get_mipmap_levels() const; __declspec(property(get=get_mipmap_levels)) unsigned int mipmap_levels;</pre> <p>Query how many mipmap levels are accessible by this texture (or texture view).</p> <p><b>Error conditions: none</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                              |              |                                                     |
| 4066         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |              |                                                     |
| 4067         | <pre>unsigned int get_data_length() const; __declspec(property(get=get_data_length)) unsigned int data_length;</pre> <p>Gets the physical data length (in bytes) that is required in order to represent the texture on the host side with its native format.</p> <p><b>Error conditions: none</b></p>                                                                                                                                                                                                                                                                                                                                                                              |              |                                                     |
| 4068         | <b>10.1.12 Querying texture's logical dimensions</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |              |                                                     |
| 4069         | <pre>extent&lt;N&gt; get_extent() const restrict(cpu,amp); __declspec(property(get=get_extent)) extent&lt;N&gt; extent;</pre> <p>These members have the same meaning as the equivalent ones on the array class</p> <p><b>Error conditions: none</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                            |              |                                                     |
| 4070         | <pre>unsigned int get_mipmap_extent(unsigned int mipmap_level) const;</pre> <p>Returns the extent for specific mipmap level of this texture (or texture view).</p> <p><b>Parameters:</b></p> <table border="1"> <tr> <td>mipmap_level</td> <td>Mipmap level for which extent should be calculated.</td> </tr> </table> <p><b>Error condition</b>    <b>Exception thrown</b></p>                                                                                                                                                                                                                                                                                                    | mipmap_level | Mipmap level for which extent should be calculated. |
| mipmap_level | Mipmap level for which extent should be calculated.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |              |                                                     |

|                               |                                |
|-------------------------------|--------------------------------|
| Invalid value of mipmap level | concurrency::runtime_exception |
|-------------------------------|--------------------------------|

4071

#### 10.1.13 Querying the accelerator\_view where the texture resides

4072

```
accelerator_view get_accelerator_view() const;
__declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
```

Retrieves the accelerator\_view where the texture resides

**Error conditions:** none

4073

```
accelerator_view get_associated_accelerator_view() const;
__declspec(property(get=get_associated_accelerator_view)) accelerator_view
associated_accelerator_view;
```

Returns the accelerator\_view that is the preferred target where this staging texture can be copied to/from.

**Error conditions:** none

4074

#### 10.1.14 Querying a staging texture's row and depth pitch

4075

```
unsigned int get_row_pitch() const;
__declspec(property(get=get_row_pitch)) unsigned int row_pitch;
```

Returns the row pitch (in bytes) of a 2D or 3D staging texture on the CPU to be used for navigating the staging texture from row to row on the CPU.

**Error conditions** Static assertion when invoked on 1D texture

4076

```
unsigned int get_depth_pitch() const;
__declspec(property(get=get_depth_pitch)) unsigned int depth_pitch;
```

Returns the depth pitch (in bytes) of a 3D staging texture on the CPU to be used for navigating the staging texture from depth slice to depth slice on the CPU.

**Error conditions** Static assertion when invoked on 1D or 2D texture

4077

#### 10.1.15 Reading and writing textures

4078

This is the core function of class texture on the accelerator. Unlike [arrays](#), the entire value type has to be get/set, and is returned or accepted wholly. [textures](#) do not support returning a reference to their data internal representation.

4079

Due to platform restrictions, only a limited number of [texture](#) types support simultaneous reading and writing. Reading is supported on all [texture](#) types, but reading and writing within same parallel\_for\_each through a [texture](#) is only supported for [textures](#) of [int](#), [uint](#), and [float](#), and even in those cases, the number of bits used in the physical format must be 32 and the number of channels should be 1. In case a lower number of bits is used (8 or 16) and a kernel is invoked which contains code that could possibly both write into and read from one of these rank-1 [texture](#) types, then an implementation is permitted to raise a runtime exception.

4080

**Microsoft-specific:** the Microsoft implementation always raises a runtime exception in such a situation.

4081

Trying to call “set” on a [texture](#) of a different element type (i.e., on other than [int](#), [uint](#), and [float](#)) results in a static assert. In order to write into [textures](#) of other value types, the developer must go through a [texture\\_view<T,N>](#).

4082

```

const value_type operator[] (const index<N>& index) const restrict(amp);
const value_type operator[] (int i0) const restrict(amp);
const value_type operator() (const index<N>& index) const restrict(amp);
const value_type operator() (int i0) const restrict(amp);
const value_type operator() (int i0, int i1) const restrict(amp);
const value_type operator() (int i0, int i1, int i2) const restrict(amp);
const value_type get(const index<N>& index) const restrict(amp);
void set(const index<N>& index, const value_type& value) const restrict(amp);

```

Loads one texel out of the texture. In case the overload where an integer tuple is used, if an overload which doesn't agree with the rank of the matrix is used, then a static\_assert ensues and the program fails to compile.

If the texture is indexed, at runtime, outside of its logical bounds, the behavior is undefined.

#### Parameters

|            |                                                                                                                                                                                                                                                              |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| index      | An N-dimension logical integer coordinate to read from                                                                                                                                                                                                       |
| i0, i1, i2 | Index components, equivalent to providing index<1>(_I0), or index<2>(i0,i1) or index<2>(i0, i1, i2). The arity of the function used must agree with the rank of the matrix. e.g., the overload which takes (i0, i1) is only available on textures of rank 2. |
| value      | Value to write into the texture                                                                                                                                                                                                                              |

**Error conditions:** if set is called on texture types which are not supported, a static\_assert ensues.

### 10.1.16 Direct3d Interop Functions

The following functions are provided in the direct3d namespace in order to convert between DX COM interfaces and textures.

4098

```

template <typename T, int N>
texture<T, N> make_texture(const Concurrency::accelerator_view& av, const IUnknown* pTexture,
DXGI_FORMAT view_format = DXGI_FORMAT_UNKNOWN);

```

Creates a texture from the corresponding DX interface. On success, it increments the reference count of the D3D texture interface by calling "AddRef" on the interface. Users must call "Release" on the returned interface after they are finished using it, for proper reclamation of the resources associated with the object.

#### Parameters

|                              |                                                                                                                                                                      |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| av                           | A D3D accelerator view on which the texture is to be created.                                                                                                        |
| pTexture                     | A pointer to a suitable texture                                                                                                                                      |
| view_format                  | The DXGI format to use for resource views created for this texture in C++ AMP kernels, or DXGI_FORMAT_UNKNOWN (the default) to use the format of the texture itself. |
| <b>Return value</b>          | Created texture                                                                                                                                                      |
| <b>Error condition</b>       | <b>Exception thrown</b>                                                                                                                                              |
| Out of memory                |                                                                                                                                                                      |
| Invalid D3D texture argument |                                                                                                                                                                      |

4099

```

template <typename T, int N>
IUnknown * get_texture<const texture<T, N>& texture>();

```

Retrieves a DX interface pointer from a C++ AMP texture object. Class texture allows retrieving a texture interface pointer (the exact interface depends on the rank of the class). On success, it increments the reference count of the D3D texture interface by calling "AddRef" on the interface. Users must call "Release" on the returned interface after they are finished using it, for proper reclamation of the resources associated with the object.

#### Parameters

|                            |                                 |
|----------------------------|---------------------------------|
| texture                    | Source texture                  |
| <b>Return value</b>        | Texture interface as IUnknown * |
| <b>Error condition: no</b> |                                 |

4100

4101 **10.2 writeonly\_texture\_view<T,N>**

4102

4103 C++ AMP write-only texture views, coded as `writeonly_texture_view<T, N>`, which provides write-only access into any  
4104 *texture*.

4105

4106 Note, `writeonly_texture_view<T, N>` is deprecated. Please use `texture_view<T, N>` instead.4107 **10.2.1 Synopsis**

4108

```
4109 template <typename T, int N>
4110 class writeonly_texture_view<T,N>
4111 {
4112 public:
4113     static const int rank = N
4114     typedef typename T value_type;
4115     typedef short_vectors_traits<T>::scalar_type scalar_type;
4116
4117     writeonly_texture_view(texture<T,N>& src) restrict(cpu,amp);
4118
4119     writeonly_texture_view(const writeonly_texture_view&) restrict(cpu,amp);
4120
4121     writeonly_texture_view operator=(const writeonly_texture_view&) restrict(cpu,amp);
4122
```

4123

```
// Microsoft-specific:
__declspec(property(get=get_bits_per_scalar_element)) int bits_per_scalar_element;
__declspec(property(get=get_data_length)) unsigned int data_length;
__declspec(property(get=get_extent)) extent<N> extent;
__declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
```

4128

```
4129     unsigned int get_bits_per_scalar_element() const;
4130     unsigned int get_data_length() const;
4131     extent<N> get_extent() const restrict(cpu,amp);
4132     accelerator_view get_accelerator_view() const;
4133
4134     void set(const index<N>& index, const value_type& val) const restrict(amp);
};
```

4135 **10.2.2 Introduced typedefs**

```
typedef ... value_type;
```

The logical value type of the `writeonly_texture_view`. e.g., for `writeonly_texture_view<float2,3>`, `value_type` would be `float2`.

4136

```
typedef ... scalar_type;
```

The scalar type that serves as the component of the texture's value type. For example, for `writeonly_texture_view<int2,3>`, the scalar type would be "int".

4137

**10.2.3 Construct a writeonly view over a texture**

```
writeonly_texture_view(texture<T,N>& src) restrict(cpu);
writeonly_texture_view(texture<T,N>& src) restrict(amp);
```

Creates a write-only view to a given texture.

When create the writeonly\_texture\_view in a restrict(amp) function, if the number of scalar elements of T is larger than 1, a compilation error will be given. A writeonly\_texture\_view cannot be created on top of staging texture.

#### Parameters

|     |                |
|-----|----------------|
| src | Source texture |
|-----|----------------|

4138

#### 10.2.4 Copy constructors and assignment operators

```
writeonly_texture_view(const writeonly_texture_view& other) restrict(cpu,amp);
writeonly_texture_view operator=(const writeonly_texture_view& other) restrict(cpu,amp);
```

writeonly\_texture\_views are shallow objects which can be copied and moved both on the CPU and on an accelerator. They are captured by value when passed to parallel\_for\_each

#### Parameters

|       |                                       |
|-------|---------------------------------------|
| other | Source writeonly_texture view to copy |
|-------|---------------------------------------|

#### Error condition

#### Exception thrown

4140

#### 10.2.5 Querying underlying texture's physical characteristics

4142

```
unsigned int get_bits_per_scalar_element() const;
__declspec(property(get=get_bits_per_scalar_element)) unsigned int bits_per_scalar_element;
```

Gets the bits-per-scalar-element of the texture

#### Error conditions: none

4143

4144

```
unsigned int get_data_length() const;
__declspec(property(get=get_data_length)) unsigned int data_length;
```

Gets the physical data length (in bytes) that is required in order to represent the texture on the host side with its native format.

#### Error conditions: none

4145

4146

```
accelerator_view get_accelerator_view() const;
__declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
```

Retrieves the accelerator\_view where the underlying texture resides.

#### Error conditions: none

4147

#### 10.2.7 Querying underlying texture's logical dimensions (through a view)

4149

```
extent<N> get_extent() const restrict(cpu,amp);
__declspec(property(get=get_extent)) extent<N> extent;
```

These members have the same meaning as the equivalent ones on the array class

#### Error conditions: none

4150

#### 10.2.8 Writing a write-only texture view

4151

This is the main purpose of this type. All *texture* types can be written through a write-only view.

4152

```
void set(const index<N>& index, const value_type& val) const restrict(amp);
```

|                               |                                                                                                                                      |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
|                               | Stores one texel in the texture.<br><br>If the texture is indexed, at runtime, outside of its logical bounds, behavior is undefined. |
| <b>Parameters</b>             |                                                                                                                                      |
| index                         | An N-dimension logical integer coordinate to read from                                                                               |
| val                           | Value to store into the texture                                                                                                      |
| <b>Error conditions:</b> none |                                                                                                                                      |

4153

4154 **10.2.9 Direct3d Interop Functions**4155 The following functions are provided in the *direct3d* namespace in order to convert between DX COM interfaces and  
4156 *writeonly\_texture\_views*.

4157

```
template <typename T, int N>
IUnknown * get_texture(const writeonly_texture_view<T, N>& texture_view);
```

Retrieves a DX interface pointer from a C++ AMP *writeonly\_texture\_view* object. On success, it increments the reference count of the D3D texture interface by calling "AddRef" on the interface. Users must call "Release" on the returned interface after they are finished using it, for proper reclamation of the resources associated with the object.

**Parameters**

|              |                     |
|--------------|---------------------|
| texture_view | Source texture view |
|--------------|---------------------|

|                     |                                 |
|---------------------|---------------------------------|
| <b>Return value</b> | Texture interface as IUnknown * |
|---------------------|---------------------------------|

|                            |
|----------------------------|
| <b>Error condition:</b> no |
|----------------------------|

4158

4159 **10.3 sampler**4160 The *sampler* class aggregates sampling configuration information, including the *filter mode*, the *addressing mode* on each  
4161 dimension of the texture, etc. Note that the constructors of this class are *restrict(cpu)* only, but its copy constructors,  
4162 assignment operators and all accessor functions are *restrict(cpu,amp)*.4163 **10.3.1 Synopsis**

```
4164
4165 class sampler
4166 {
4167 public:
4168     sampler();
4169     sampler(filter_mode filter_mode);
4170     sampler(address_mode address_mode, float_4 aorder_color=float_4(0.0f, 0.0f, 0.0f, 0.0f));
4171     sampler(filter_mode filter_mode, address_mode address_mode,
4172             float_4 border_color=float_4(0.0f, 0.0f, 0.0f, 0.0f));
4173
4174     sampler(const sampler& other) restrict(cpu,amp);
4175     sampler(sampler&& other) restrict(cpu,amp);
4176
4177     sampler& operator=(const sampler& other) restrict(cpu,amp);
4178     sampler& operator=(sampler&& other) restrict(cpu,amp);
4179
4180     // Microsoft-specific:
4181     __declspec(property(get=get_filter_mode)) filter_mode filter_mode;
4182     __declspec(property(get=get_address_mode)) address_mode address_mode;
4183     __declspec(property(get=get_border_color)) float_4 border_color;
```

4184

```

4185     filter_mode get_filter_mode() const restrict(cpu,amp);
4186     address_mode get_address_mode() const restrict(cpu,amp);
4187     float_4 get_border_color() const restrict(cpu,amp);
4188 };

```

### 4189 10.3.2 filter\_modes

```

4190
4191 enum filter_mode
4192 {
4193     filter_point,
4194     filter_linear,
4195     filter_unknown
4196 };
4197

```

4198 This enumeration is used to specify the filter mode of a [sampler](#). It controls what and how texels are read and combined to  
 4199 produce interpolated values during sampling. Currently only two filter modes are exposed in C++ AMP APIs which  
 4200 correspond to the two simplest and most common filter modes, that is, the filters used for minification, magnification and  
 4201 mip level sampling are all same, either all point or all linear. *filter\_unknown* represents filter modes that are not exposed by  
 4202 C++ AMP APIs, but are adopted from the underlying platform.  
 4203

4204 ***Microsoft-specific:*** The two filter modes exposed by C++ AMP corresponds to DirectX filter enum (*D3D11\_FILTER*):  
 4205 *D3D11\_FILTER\_MIN\_MAG\_MIP\_POINT* and *D3D11\_FILTER\_MIN\_MAG\_MIP\_LINEAR* respectively. The Microsoft  
 4206 implementation of C++ AMP sets the enum values to be same as DirectX corresponding enum values for efficient interop  
 4207 support.

### 4208 10.3.3 address\_mode

```

4209
4210 enum address_mode
4211 {
4212     address_wrap,
4213     address_mirror,
4214     address_clamp,
4215     address_border,
4216     address_unknown
4217 };

```

4218 This enumeration is used to specify the addressing mode of a sampler. It controls how sampling handles texture  
 4219 coordinates that are outside of the boundaries of a texture. Texture's normalized coordinates are always between the  
 4220 range of 0.0 to 1.0 inclusive. The addressing mode of the texture determines how to map out-of-range coordinates to its  
 4221 normalized domain, which could be used to generate special effects of texture mapping. *address\_unknown* represents  
 4222 address modes that are not exposed by C++ AMP APIs, but are adopted from the underlying platform.  
 4223

4224 ***Microsoft-specific:***  
 4225 The Microsoft implementation of C++ AMP sets the enum values to be same as DirectX corresponding enum values for  
 4226 efficient interop support.

### 4227 10.3.4 Constructors

```
4228
```

```

sampler();
sampler(filter_mode filter_mode);
sampler(address_mode address_mode, float_4 border_color=float_4(0.0f, 0.0f, 0.0f, 0.0f));
sampler(filter_mode filter_mode, address_mode address_mode,

```

|                                                                                                                                                                                                                               |                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
|                                                                                                                                                                                                                               | <pre>float_4 border_color=float_4(0.0f, 0.0f, 0.0f, 0.0f));</pre>      |
| Constructs a sampler with specified filter mode (same for min, mag, mip), addressing mode (same for all dimensions) and the border color.                                                                                     |                                                                        |
| <b>Parameters</b>                                                                                                                                                                                                             |                                                                        |
| <i>filter_mode</i>                                                                                                                                                                                                            | The filter mode to be used in sampling.                                |
| <i>faddress_mode</i>                                                                                                                                                                                                          | The addressing mode of all dimensions of the texture.                  |
| <i>border_color</i>                                                                                                                                                                                                           | The border color to be used if address mode is <i>address_border</i> . |
| The following default values are used when a parameter is not specified:<br><i>filter_mode</i> <i>filter_linear</i><br><i>address_mode</i> <i>address_clamp</i><br><i>border_color</i> <i>float_4(0.0f, 0.0f, 0.0f, 0.0f)</i> |                                                                        |
| <b>Error condition</b>                                                                                                                                                                                                        | <b>Exception thrown</b>                                                |
| Out of memory                                                                                                                                                                                                                 | concurrency::runtime_exception                                         |
| Unknown filter mode or address mode                                                                                                                                                                                           | concurrency::runtime_exception                                         |

4229

|                                                                 |
|-----------------------------------------------------------------|
| <pre>sampler(const sampler&amp; other) restrict(cpu,amp);</pre> |
|-----------------------------------------------------------------|

Copy constructor. Constructs a new sampler object from the supplied argument other.

**Parameters**

|              |                                                                      |
|--------------|----------------------------------------------------------------------|
| <i>other</i> | An object of type sampler from which to initialize this new sampler. |
|--------------|----------------------------------------------------------------------|

4230

|                                                                |
|----------------------------------------------------------------|
| <pre>sampler(sampler&amp;&amp; other) restrict(cpu,amp);</pre> |
|----------------------------------------------------------------|

Move constructor.

**Parameters**

|              |                                         |
|--------------|-----------------------------------------|
| <i>other</i> | An object of type sampler to move from. |
|--------------|-----------------------------------------|

4231

### 10.3.5 Members

4232

|                                                                                |
|--------------------------------------------------------------------------------|
| <pre>sampler&amp; operator=(const sampler&amp; other) restrict(cpu,amp);</pre> |
|--------------------------------------------------------------------------------|

Assignment operator. Assigns the contents of the sampler object “\_Other” to “this” sampler object and returns a reference to “this” object.

**Parameters**

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| <i>other</i> | An object of type sampler from which to copy into this sampler. |
|--------------|-----------------------------------------------------------------|

4233

|                                                                               |
|-------------------------------------------------------------------------------|
| <pre>sampler&amp; operator=(sampler&amp;&amp; other) restrict(cpu,amp);</pre> |
|-------------------------------------------------------------------------------|

Move assignment operator.

**Parameters**

|              |                                         |
|--------------|-----------------------------------------|
| <i>other</i> | An object of type sampler to move from. |
|--------------|-----------------------------------------|

4234

|                                                                                           |
|-------------------------------------------------------------------------------------------|
| <pre>_declspec(property(get=get_filter_mode)) Concurrency::filter_mode filter_mode;</pre> |
|-------------------------------------------------------------------------------------------|

|                                                                                |
|--------------------------------------------------------------------------------|
| <pre>Concurrency::filter_mode get_filter_mode() const restrict(cpu,amp);</pre> |
|--------------------------------------------------------------------------------|

Access the filter mode.

4235

|                                                                            |
|----------------------------------------------------------------------------|
| <pre>_declspec(property(get=get_border_color)) float_4 border_color;</pre> |
|----------------------------------------------------------------------------|

|                                                                |
|----------------------------------------------------------------|
| <pre>float_4 get_border_color() const restrict(cpu,amp);</pre> |
|----------------------------------------------------------------|

4236

4237

Access the border color.

4238

4239

```
__declspec(property(get=get_address_mode)) Concurrency::address_mode address_mode;
Concurrency::address_mode get_address_mode() const restrict(cpu,amp);
```

Access the addressing mode.

4240

#### 4241 10.3.6 Direct3d Interop Functions

4242

4243 The following functions are provided in the *direct3d* namespace in order to convert between DX COM interfaces and  
 4244 sampler objects.

4245

```
sampler make_sampler(const IUnknown* D3D_sampler);
```

Adopt a sampler from the corresponding DX sampler state interface.

**Parameters**

|                    |                                                      |
|--------------------|------------------------------------------------------|
| <i>D3D_sampler</i> | A pointer to a suitable D3D sampler-state interface. |
|--------------------|------------------------------------------------------|

|                     |                            |
|---------------------|----------------------------|
| <b>Return value</b> | The adopted sampler object |
|---------------------|----------------------------|

|                        |                         |
|------------------------|-------------------------|
| <b>Error condition</b> | <b>Exception thrown</b> |
|------------------------|-------------------------|

|               |                                |
|---------------|--------------------------------|
| Out of memory | concurrency::runtime_exception |
|---------------|--------------------------------|

|                                    |                                |
|------------------------------------|--------------------------------|
| Invalid D3D sampler-state argument | concurrency::runtime_exception |
|------------------------------------|--------------------------------|

4246

```
IUnknown* get_sampler(const Concurrency::accelerator_view& av, const sampler& sampler);
```

Get the D3D sampler state interface on the given accelerator view that represents the specified sampler object.

**Parameters**

|           |                                                                             |
|-----------|-----------------------------------------------------------------------------|
| <i>av</i> | A D3D accelerator view on which the D3D sampler state interface is created. |
|-----------|-----------------------------------------------------------------------------|

|                |                                                                                   |
|----------------|-----------------------------------------------------------------------------------|
| <i>sampler</i> | A sampler object for which the underlying D3D sampler state interface is created. |
|----------------|-----------------------------------------------------------------------------------|

|                     |                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------|
| <b>Return value</b> | The IUnknown interface pointer corresponding to the D3D sampler state that represents the given sampler. |
|---------------------|----------------------------------------------------------------------------------------------------------|

4247

#### 4248 10.4 texture\_view<T,N>

4249

4250 The *texture\_view<T, N>* class provides read-write access on top of textures. It is bound to a specific mipmap level of the underlying texture object.

4251

4252

```
template <typename T, int N>
class texture_view<T,N>
{
public:
    static const int rank = N;
    typedef T value_type;
    typedef typename short_vectors_traits<T>::scalar_type scalar_type;

    texture_view(texture<T,N>&, unsigned int mipmap_level = 0) restrict(cpu);
    texture_view(texture<T,N>&) restrict(amp);

    texture_view(const texture_view& other) restrict(cpu,amp);
    texture_view operator=(const texture_view& other) restrict(cpu,amp);
```

```

4266 // Microsoft-specific:
4267 __declspec(property(get=get_bit_per_scalar_element)) unsigned int bits_per_scalar_element;
4268 __declspec(property(get=get_mipmap_levels)) unsigned int mipmap_levels;
4269 __declspec(property(get=get_data_length)) unsigned int data_length;
4270 __declspec(property(get=get_extent)) extent<N> extent;
4271 __declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;

4272     unsigned int get_bits_per_scalar_element() const;
4273     unsigned int get_mipmap_levels() const;
4274     unsigned int get_data_length() const;
4275     extent<N> get_extent() const restrict(cpu,amp);
4276     extent<N> get_mipmap_extent(unsigned int mipmap_level) const restrict(cpu,amp);
4277     accelerator_view get_accelerator_view() const;
4278
4279     const value_type operator[] (const index<N>& index) const restrict(amp);
4280     const value_type operator[] (int i0) const restrict(amp);
4281     const value_type operator() (const index<N>& index) const restrict(amp);
4282     const value_type operator() (int i0) const restrict(amp);
4283     const value_type operator() (int i0, int i1) const restrict(amp);
4284     const value_type operator() (int i0, int i1, int i2) const restrict(amp);
4285     const value_type get(const index<N>& index) const restrict(amp);
4286
4287     void set(const index<N>& index, const value_type& val) const restrict(amp);
4288 };
4289

```

#### 4290 10.4.2 Introduced typedefs

4291

|                                      |
|--------------------------------------|
| <code>typedef ... value_type;</code> |
|--------------------------------------|

The logical value type of the texture\_view. e.g., for `texture_view<float2,3>`, `value_type` would be `float2`.

4292

|                                       |
|---------------------------------------|
| <code>typedef ... scalar_type;</code> |
|---------------------------------------|

The scalar type that serves as the component of the texture's value type. For example, for `texture_view<int2,3>`, the scalar type would be "int".

4293

#### 4294 10.4.3 Constructors

4295

|                                                                                                      |
|------------------------------------------------------------------------------------------------------|
| <code>texture_view(texture&lt;T,N&gt;&amp; src, unsigned int mipmap_level = 0) restrict(cpu);</code> |
|------------------------------------------------------------------------------------------------------|

Creates a texture view to a given mipmap level of a texture on host. The source texture cannot be a staging texture.

**Parameters**

|     |                |
|-----|----------------|
| src | Source texture |
|-----|----------------|

|              |                                                          |
|--------------|----------------------------------------------------------|
| mipmap_level | The mipmap this view represents, the default value is 0. |
|--------------|----------------------------------------------------------|

|                        |                         |
|------------------------|-------------------------|
| <b>Error condition</b> | <b>Exception thrown</b> |
|------------------------|-------------------------|

|                        |                                |
|------------------------|--------------------------------|
| src is staging texture | concurrency::runtime_exception |
|------------------------|--------------------------------|

4296

|                                                                       |
|-----------------------------------------------------------------------|
| <code>texture_view(texture&lt;T,N&gt;&amp; src) restrict(amp);</code> |
|-----------------------------------------------------------------------|

Creates a texture view to a given texture on accelerator on the most detailed mipmap level.

**Parameters**

|     |                |
|-----|----------------|
| src | Source texture |
|-----|----------------|

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>Error condition</b> | When create the <code>texture_view</code> in a <code>restrict(amp)</code> function, if the number of scalar elements of |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|

|  |                                                        |
|--|--------------------------------------------------------|
|  | T is larger than 1, a compilation error will be given. |
|--|--------------------------------------------------------|

4297

4298 **10.4.4 Copy constructors and assignment operators**

4299

```
texture_view(const texture_view& other) restrict(cpu,amp);
texture_view operator=(const texture_view& other) restrict(cpu,amp);
```

texture\_views are shallow objects which can be copied and moved both on the CPU and on an accelerator. They are captured by value when passed to parallel\_for\_each

**Parameters**

|       |                                  |
|-------|----------------------------------|
| other | Source texture view to copy from |
|-------|----------------------------------|

**Error conditions: none**4300 **10.4.5 Query functions**4301 **10.4.5.1 Querying texture's physical characteristics**

4302

```
unsigned int get_bits_per_scalar_element() const;
__declspec(property(get=get_bits_per_scalar_element)) unsigned int bits_per_scalar_element;
```

Gets the bits-per-scalar-element of the texture.

**Microsoft-specific:** Returns 0, if the texture is created using Direct3D Interop (10.1.16).

**Error conditions: none**

4303

```
unsigned int get_mipmap_levels() const;
__declspec(property(get=get_mipmap_levels)) unsigned int mipmap_Levels;
```

Query how many mipmap levels are accessible by this texture view. This will always return 1 as texture\_view<T, N> is bound to a single mipmap level at a time.

**Error conditions: none**

4304

```
unsigned int get_data_length() const;
__declspec(property(get=get_data_length)) unsigned int data_Length;
```

Gets the physical data length (in bytes) that is required in order to represent the texture on the host side with its native format.

**Error conditions: none**4305 **10.4.5.2 Querying texture's logical dimensions**

4306

```
extent<N> get_extent() const restrict(cpu,amp);
__declspec(property(get=get_extent)) extent<N> extent;
```

These members have the same meaning as the equivalent ones on the array class

**Error conditions: none**

4307

```
extent<N> get_mipmap_extent(unsigned int mipmap_level) const restrict(cpu,amp);
```

Returns the extent for specific mipmap level of this texture view. The behavior is undefined for invalid value of mipmap level when invoked in amp restricted context.

**Parameters:**

|              |                                                     |
|--------------|-----------------------------------------------------|
| mipmap_level | Mipmap level for which extent should be calculated. |
|--------------|-----------------------------------------------------|

| Error conditions               | Exception thrown               |
|--------------------------------|--------------------------------|
| Invalid value for mipmap level | Concurrency::runtime_exception |

4308

## 4309 10.4.5.3 Querying the accelerator\_view where the texture resides

4310

```
accelerator_view get_accelerator_view() const;
__declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
```

Retrieves the accelerator\_view where the texture resides.

**Error conditions:** none

## 4311 10.4.6 Reading and writing a texture\_view

4312

4313 Only a limited number of *texture\_view* types support simultaneous reading and writing. Writing is supported on all  
 4314 *texture\_view* types, but reading through a *texture\_view* is only supported for *texture\_views* of *int*, *uint*, and *float*, and even  
 4315 in those cases, the number of bits used in the physical format must be 32. In case a lower number of bits is used (8 or 16)  
 4316 and a kernel is invoked which contains code that could possibly both write into and read from one of these rank-1  
 4317 *texture\_view* types, then an implementation is permitted to raise a runtime exception.

4318

4319 **Microsoft-specific:** the Microsoft implementation always raises a runtime exception in such a situation.

4320

```
const value_type operator[] (const index<N>& index) const restrict(amp);
const value_type operator[] (int i0) const restrict(amp);
const value_type operator() (const index<N>& index) const restrict(amp);
const value_type operator() (int i0) const restrict(amp);
const value_type operator() (int i0, int i1) const restrict(amp);
const value_type operator() (int i0, int i1, int i2) const restrict(amp);
const value_type get(const index<N>& index) const restrict(amp);
```

Loads one texel out of the *texture\_view*. In case of the overload where an integer tuple is used, if an overload doesn't agree with the rank of the *texture\_view*, then a static\_assert ensues and the program fails to compile.

If the underlying texture is indexed outside of its logical bounds at runtime, behavior is undefined

Trying to read on a *texture\_view* of a value type other than *int*, *uint*, and *float* results in a static assert. In order to read from *texture\_views* of other value types, the developer must go through a *texture\_view<const T,N>*.

**Parameters**

|            |                                                                                                                                                                                                                                                                                                       |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| index      | An N-dimension logical integer coordinate to read from                                                                                                                                                                                                                                                |
| i0, i1, i0 | Index components, equivalent to providing <i>index&lt;1&gt;(i0)</i> , or <i>index&lt;2&gt;(i0, i1)</i> or <i>index&lt;2&gt;(i0, i1, i2)</i> . The arity of the function used must agree with the rank of the matrix. e.g., the overload which takes (i0, i1) is only available on textures of rank 2. |

**Error conditions:** If these methods are called on *texture\_view* types which are not supported, a static\_assert ensues.

4321

4322

```
void set(const index<N>& index, const value_type& val) const restrict(amp);
```

Stores one texel in the underlying texture represented by the *texture\_view*

If the underlying texture is indexed, at runtime, outside of its logical bounds, behavior is undefined.

**Parameters**

|       |                                                        |
|-------|--------------------------------------------------------|
| index | An N-dimension logical integer coordinate to read from |
|-------|--------------------------------------------------------|

|                               |                                 |
|-------------------------------|---------------------------------|
| val                           | Value to store into the texture |
| <b>Error conditions:</b> none |                                 |

4323

4324 **10.4.7 Direct3d Interop Functions**4325 The following functions are provided in the *direct3d* namespace in order to convert between DirectX COM interfaces and  
4326 *texture\_view*.

4327

```
template <typename T, int N>
IUnknown * get_texture(const texture_view<T, N>& texture_view);
```

Retrieves a DX interface pointer from a C++ AMP writeonly\_texture\_view object. On success, it increments the reference count of the D3D texture interface by calling "AddRef" on the interface. Users must call "Release" on the returned interface after they are finished using it, for proper reclamation of the resources associated with the object.

**Parameters**

|              |                     |
|--------------|---------------------|
| texture_view | Source texture view |
|--------------|---------------------|

|                     |                                 |
|---------------------|---------------------------------|
| <b>Return value</b> | Texture interface as IUnknown * |
|---------------------|---------------------------------|

|                            |
|----------------------------|
| <b>Error condition:</b> no |
|----------------------------|

4328 **10.5 texture\_view<const T,N>**4329 The *texture\_view<const T, N>* class provides a read-only access and richer data load functionality on top of textures. It  
4330 exposes special features of the graphics hardware useful in rendering and image processing, such as texture sampling and  
4331 gathering, and the ability to load values from multiple mipmap levels in the same kernel.4332 **10.5.1 Synopsis**

```
template <typename T, int N>
class texture_view<const T, N>
{
public:
    static const int rank = N;
    typedef const T value_type;
    typedef typename short_vectors_traits<T>::scalar_type scalar_type;
    typedef typename short_vector<float,N>::type coordinates_type;
    typedef typename short_vector<scalar_type,4>::type gather_return_type;

    texture_view(const texture<T,N>& src) restrict(cpu);
    texture_view(const texture<T,N>& src, unsigned int most_detailed_mip, unsigned int
mip_levels) restrict(cpu);

    texture_view(const texture<T,N>& src) restrict(amp);

    texture_view(const texture_view<T,N>& other);

    texture_view(const texture_view<const T,N>& other) restrict(cpu,amp);
    texture_view(const texture_view<const T,N>& other, unsigned int most_detailed_mip, unsigned
int mip_levels) restrict(cpu);

    texture_view operator=(const texture_view<const T,N>& other) restrict(cpu,amp);
    texture_view operator=(const texture_view<T,N>& other) restrict(cpu);

// Microsoft-specific:
__declspec(property(get=get_bit_per_scalar_element)) unsigned int bits_per_scalar_element;
__declspec(property(get=get_mipmap_levels)) unsigned int mipmap_levels;
__declspec(property(get=get_data_length)) unsigned int data_length;
```

```

4363     __declspec(property(get=get_extent)) extent<N> extent;
4364     __declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
4365
4366     unsigned int get_bits_per_scalar_element() const;
4367     unsigned int get_mipmap_levels() const;
4368     unsigned int get_data_length() const;
4369     extent<N> get_extent() const restrict(cpu,amp);
4370     extent<N> get_mipmap_extent(unsigned int mipmap_level) const restrict(cpu,amp);
4371     accelerator_view get_accelerator_view() const;
4372
4373     value_type operator[] (const index<N>& index) const restrict(amp);
4374     value_type operator[] (int i0) const restrict(amp);
4375     value_type operator() (const index<N>& index) const restrict(amp);
4376     value_type operator() (int i0) const restrict(amp);
4377     value_type operator() (int i0, int i1) const restrict(amp);
4378     value_type operator() (int i0, int i1, int i2) const restrict(amp);
4379     value_type get(const index<N>& index, unsigned int mip_level = 0) const restrict(amp);
4380
4381     value_type sample(const sampler& sampler, const coordinates_type& coord, float
4382 level_of_detail = 0.0f) const restrict(amp);
4383
4384     template<filter_mode filter_mode = filter_linear, address_mode address_mode = address_clamp>
4385     value_type sample(const coordinates_type& coord, float level_of_detail = 0.0f) const
4386     restrict(amp);
4387
4388     const gather_return_type gather_red(const sampler& sampler,
4389   const coordinates_type& coord) const restrict(amp);
4390     const gather_return_type gather_green(const sampler& sampler,
4391   const coordinates_type& coord) const restrict(amp);
4392     const gather_return_type gather_blue(const sampler& sampler,
4393   const coordinates_type& coord) const restrict(amp);
4394     const gather_return_type gather_alpha(const sampler& sampler,
4395   const coordinates_type& coord) const restrict(amp);
4396
4397     template<address_mode address_mode = address_clamp>
4398     const gather_return_type gather_red(const coordinates_type& coord) const restrict(amp);
4399
4400     template<address_mode address_mode = address_clamp>
4401     const gather_return_type gather_green(const coordinates_type& coord) const restrict(amp);
4402
4403     template<address_mode address_mode = address_clamp>
4404     const gather_return_type gather_blue(const coordinates_type& coord) const restrict(amp);
4405
4406     template<address_mode address_mode = address_clamp>
4407     const gather_return_type gather_alpha(const coordinates_type& coord) const restrict(amp);
4408 };

```

#### 4409 10.5.2 Introduced typedefs

4410

```
typedef ... value_type;
```

The logical value type of the readonly texture\_view. e.g., for texture\_view<const float2,3>, value\_type would be "const float2".

4411

```
typedef ... scalar_type;
```

The scalar type that serves as the component of the texture's value type. For example, for `texture_view<const int2,3>`, the scalar type would be "int".

4412

```
typedef ... coordinates_type;
```

The coordinates type that is used to index into the texture when sampling it. It is a short float vector whose rank is the same as that of the `texture_view`. For example, for `texture_view<const AnyT,3>`, the coordinates type will be `float3`.

4413

```
typedef ... gather_return_type;
```

The return type of gathering functions. It is a rank 4 short vector type whose scalar type is same as that of the `texture_view`. For example, for `texture_view<const float2,3>`, the `gather_return_type` will be `float4`.

4414

### 10.5.3 Constructors

4415

```
texture_view(const texture<T,N>& src) restrict(cpu);
```

Constructs a readonly view over a texture on host. The source texture cannot be a staging texture.

**Parameters**

|     |                                                    |
|-----|----------------------------------------------------|
| src | The texture where the readonly view is created on. |
|-----|----------------------------------------------------|

**Error condition**
**Exception thrown**

|                        |                                |
|------------------------|--------------------------------|
| src is staging texture | concurrency::runtime_exception |
|------------------------|--------------------------------|

4416

```
texture_view(const texture<T,N>& src) restrict(amp);
```

Constructs a readonly view over a texture on accelerator. The source texture cannot be a staging texture.

When create the `texture_view<const T,N>` in a `restrict(amp)` function, if the number of scalar elements of T is 1, a compilation error will be given.

**Parameters**

|     |                                                    |
|-----|----------------------------------------------------|
| src | The texture where the readonly view is created on. |
|-----|----------------------------------------------------|

|                        |                                                                                                                                                                                    |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Error condition</b> | When create the <code>texture_view&lt;const T,N&gt;</code> in a <code>restrict(amp)</code> function, if the number of scalar elements of T is 1, a compilation error will be given |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

4417

```
texture_view(const texture<T,N>& src, unsigned int most_detailed_mip = 0, unsigned int mip_levels = 1) restrict(cpu);
```

Constructs a readonly view over a texture on host. The source texture cannot be a staging texture.

**Parameters**

|     |                                                    |
|-----|----------------------------------------------------|
| src | The texture where the readonly view is created on. |
|-----|----------------------------------------------------|

|                   |                                                                  |
|-------------------|------------------------------------------------------------------|
| most_detailed_mip | Sets the most detailed mip for the view, the default value is 0. |
|-------------------|------------------------------------------------------------------|

|            |                                                                                                                                   |
|------------|-----------------------------------------------------------------------------------------------------------------------------------|
| mip_levels | The number of mip levels viewable by the texture view starting from <code>most_detailed_mip</code> level. The default value is 1. |
|------------|-----------------------------------------------------------------------------------------------------------------------------------|

**Error condition**
**Exception thrown**

|                        |                                |
|------------------------|--------------------------------|
| src is staging texture | concurrency::runtime_exception |
|------------------------|--------------------------------|

|                                  |                                |
|----------------------------------|--------------------------------|
| Invalid values for mipmap levels | concurrency::runtime_exception |
|----------------------------------|--------------------------------|

4418

```
texture_view(const texture_view<T,N>& src);
```

Constructs a readonly `texture_view` over a writable `texture_view`.

**Parameters**

|     |                                                                      |
|-----|----------------------------------------------------------------------|
| src | The <code>texture_view</code> where the readonly view is created on. |
|-----|----------------------------------------------------------------------|

4420

4421 **10.5.4 Copy constructors and assignment operators**

4422

```
texture_view(const texture_view<const T,N>& other) restrict(cpu,amp);
texture_view operator=(const texture_view<const T,N>& other) restrict(cpu,amp);
texture_view operator=(const texture_view<T,N>& other) restrict(cpu);
```

texture\_views are shallow objects which can be copied and moved both on the CPU and on an accelerator. They are captured by value when passed to parallel\_for\_each.

**Parameters**

|       |                                   |
|-------|-----------------------------------|
| other | Source texture_view to copy from. |
|-------|-----------------------------------|

4423

```
texture_view(const texture_view<const T,N>& src, unsigned int most_detailed_mip = 0, unsigned
int mip_levels = 1) restrict(cpu);
```

Constructs a readonly texture\_view over a readonly texture\_view.

**Parameters**

|                   |                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------|
| src               | The texture_view where the readonly view is created on.                                                                   |
| most_detailed_mip | Sets the most detailed mip for the view, the default value is 0. The value is relative to the source's most detailed mip. |
| mip_levels        | The number of mip levels viewable by the texture view starting from most_detailed_mip level.                              |

**Error condition****Exception thrown**

|                                  |                                |
|----------------------------------|--------------------------------|
| Invalid values for mipmap levels | Concurrency::runtime_exception |
|----------------------------------|--------------------------------|

4424

4425 **10.5.5 Query functions**4426 **10.5.5.1 Querying texture's physical characteristics**

4427

```
unsigned int get_bits_per_scalar_element() const;
__declspec(property(get=get_bits_per_scalar_element)) unsigned int bits_per_scalar_element;
```

Gets the bits-per-scalar-element of the texture. Returns 0, if the texture is created using Direct3D Interop (10.1.16).

**Error conditions: none**

4428

```
unsigned int get_mipmap_levels() const;
__declspec(property(get=get_mipmap_levels)) unsigned int mipmap_levels;
```

Query how many mipmap levels are accessible by this texture view.

**Error conditions: none**

4429

4430

```
unsigned int get_data_length() const;
__declspec(property(get=get_data_length)) unsigned int data_length;
```

Gets the physical data length (in bytes) that is required in order to represent the texture on the host side with its native format.

**Error conditions: none**

4431

**10.5.5.2 Querying texture's logical dimensions**

4432

```
extent<N> get_extent() const restrict(cpu,amp);
__declspec(property(get=get_extent)) extent<N> extent;
```

These members have the same meaning as the equivalent ones on the array class

**Error conditions:** none

4433

```
extent<N> get_mipmap_extent(unsigned int mipmap_level) const restrict(cpu,amp);
```

Returns the extent for specific mipmap level of this texture view. The behavior is undefined for invalid value of mipmap level when invoked in amp restricted context.

**Parameters:**

|              |                                                     |
|--------------|-----------------------------------------------------|
| mipmap_level | Mipmap level for which extent should be calculated. |
|--------------|-----------------------------------------------------|

|                        |                         |
|------------------------|-------------------------|
| <b>Error condition</b> | <b>Exception thrown</b> |
|------------------------|-------------------------|

|                                |                                |
|--------------------------------|--------------------------------|
| Invalid value for mipmap level | Concurrency::runtime_exception |
|--------------------------------|--------------------------------|

4434

4435 10.5.5.3 Querying the accelerator\_view where the texture resides

4436

```
accelerator_view get_accelerator_view() const;
__declspec(property(get=get_accelerator_view)) accelerator_view accelerator_view;
```

Retrieves the accelerator\_view where the texture resides

**Error conditions:** none

4437 10.5.6 Indexing operations

4438

```
value_type operator[] (const index<N>& index) const restrict(amp);
value_type operator[] (int i0) const restrict(amp);
value_type operator() (const index<N>& index) const restrict(amp);
value_type operator() (int i0) const restrict(amp);
value_type operator() (int i0, int i1) const restrict(amp);
value_type operator() (int i0, int i1, int i2) const restrict(amp);
value_type get(const index<N>& index, unsigned int mip_level = 0) const restrict(amp);
```

Loads one texel out of the underlying texture represented by the readonly texture\_view. In case the overload where an integer tuple is used, if an overload which doesn't agree with the rank of the texture\_view, then a static\_assert ensues and the program fails to compile.

If the underlying texture is indexed outside of its logical bounds or wrong mipmap level is supplied at runtime, behavior is undefined.

**Parameters**

|       |                                                        |
|-------|--------------------------------------------------------|
| index | An N-dimension logical integer coordinate to read from |
|-------|--------------------------------------------------------|

|            |                                                                                                                                                                                                                                                              |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| i0, i1, i0 | Index components, equivalent to providing index<1>(i0), or index<2>(i0, i1) or index<2>(i0, i1, i2). The arity of the function used must agree with the rank of the matrix. e.g., the overload which takes (i0, i1) is only available on textures of rank 2. |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|           |                                           |
|-----------|-------------------------------------------|
| mip_level | A mip level from which to read the texel. |
|-----------|-------------------------------------------|

**Error conditions:** none

4439

4440 10.5.7 Sampling operations

4441

4442 This is one of the core functionalities of the texture\_view<const T,N> type. Note that sampling is only supported when

4443 *value\_type* is based on a floating point type (i.e., float, norm or unorm). Invoking sampling operations on non-supported

4444 texture formats results in a static\_assert.

4445

4446 If address\_mode is address\_border, the named components of the sampler's border\_color are used to set values of  
 4447 corresponding named components of the returned object. For example, if the texture\_view's value\_type is float2, its x  
 4448 component takes the value of the border color's x component, and its y component takes the value of the border color's y  
 4449 component. If the value\_type is normalized float type (norm, unorm, etc.), the value of border color's components are  
 4450 clamped to the range of [-1.0, 1.0] for norm and [0.0, 1.0] for unorm.

4451

```
value_type sample(const sampler& sampler,
                 const coordinates_type& coord, float level_of_detail = 0.0f) const restrict(amp);
```

Sample the texture at the given coordinates using the specified sampling configuration.

#### Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| sampler | The sampler that configures the sampling operation |
|---------|----------------------------------------------------|

|       |                                |
|-------|--------------------------------|
| coord | Coordinate vector for sampling |
|-------|--------------------------------|

|                 |                                      |
|-----------------|--------------------------------------|
| level_of_detail | Defines the mip level from to sample |
|-----------------|--------------------------------------|

|                     |                        |
|---------------------|------------------------|
| <b>Return value</b> | The interpolated value |
|---------------------|------------------------|

4452

```
template<filter_mode filter_mode = filter_linear, address_mode address_mode = address_clamp>
value_type sample(const coordinates_type& coord, float level_of_detail = 0.0f) const
restrict(amp);
```

Sampling with predefined sampler objects based on the specified template arguments.

#### Template parameters

|             |                                                   |
|-------------|---------------------------------------------------|
| filter_mode | The filter mode of the predefined sampler to use. |
|-------------|---------------------------------------------------|

|              |                                                                                      |
|--------------|--------------------------------------------------------------------------------------|
| address_mode | The address mode of the predefined sampler to use. static_assert with address_border |
|--------------|--------------------------------------------------------------------------------------|

#### Parameters

|       |                                |
|-------|--------------------------------|
| coord | Coordinate vector for sampling |
|-------|--------------------------------|

|                 |                                      |
|-----------------|--------------------------------------|
| level_of_detail | Defines the mip level from to sample |
|-----------------|--------------------------------------|

|                     |                        |
|---------------------|------------------------|
| <b>Return value</b> | The interpolated value |
|---------------------|------------------------|

4453

### 10.5.8 Gathering operations

4455

4456 Gather operations fetch a specific component of texel values from the four points being sampled and return them all in a  
 4457 rank 4 short vector type. Note that only the addressing modes of the sampler are used. The four samples that would  
 4458 contribute to filtering are placed into xyzw of the returned value in counter clockwise order starting with the sample to the  
 4459 lower left of the queried location.

4460

4461 Note that gathering is only supported for 2D texture whose value\_type is based on a floating point type. Invoking gathering  
 4462 operations on non-supported texture formats results in a static\_assert.  
 4463

```
const gather_return_type gather_red(const sampler& sampler,
                                    const coordinates_type& coord) const restrict(amp);
```

Gathers the red component of all four samples around a sample coordinate on the mip level 0 (most detailed level) of  
 mipmap represented by the texture\_view<const T,N>.

#### Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| sampler | The sampler that configures the sampling operation |
|---------|----------------------------------------------------|

|       |                          |
|-------|--------------------------|
| coord | Coordinate for sampling. |
|-------|--------------------------|

|                     |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <b>Return value</b> | Rank 4 short vector containing the red component of the 4 texel values sampled. |
|---------------------|---------------------------------------------------------------------------------|

4464

```
const gather_return_type gather_green(const sampler& sampler,
                                     const coordinates_type& coord) const restrict(amp);
```

Gathers the green component of all four samples around a sample coordinate on the mip level 0 (most detailed level) of mipmaps represented by the texture\_view<const T,N>.

#### Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| sampler | The sampler that configures the sampling operation |
|---------|----------------------------------------------------|

|       |                          |
|-------|--------------------------|
| coord | Coordinate for sampling. |
|-------|--------------------------|

|                     |                                                                                   |
|---------------------|-----------------------------------------------------------------------------------|
| <b>Return value</b> | Rank 4 short vector containing the green component of the 4 texel values sampled. |
|---------------------|-----------------------------------------------------------------------------------|

4465

```
const gather_return_type gather_blue(const sampler& sampler,
                                     const coordinates_type& coord) const restrict(amp);
```

Gathers the blue component of all four samples around a sample coordinate on the mip level 0 (most detailed level) of mipmaps represented by the texture\_view<const T,N>.

#### Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| sampler | The sampler that configures the sampling operation |
|---------|----------------------------------------------------|

|       |                          |
|-------|--------------------------|
| coord | Coordinate for sampling. |
|-------|--------------------------|

|                     |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <b>Return value</b> | Rank 4 short vector containing the blue component of the 4 texel values sampled. |
|---------------------|----------------------------------------------------------------------------------|

4466

```
const gather_return_type gather_alpha(const sampler& sampler,
                                      const coordinates_type& coord) const restrict(amp);
```

Gathers the alpha component of all four samples around a sample coordinate on the mip level 0 (most detailed level) of mipmaps represented by the texture\_view<const T,N>.

#### Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| sampler | The sampler that configures the sampling operation |
|---------|----------------------------------------------------|

|       |                          |
|-------|--------------------------|
| coord | Coordinate for sampling. |
|-------|--------------------------|

|                     |                                                                                   |
|---------------------|-----------------------------------------------------------------------------------|
| <b>Return value</b> | Rank 4 short vector containing the alpha component of the 4 texel values sampled. |
|---------------------|-----------------------------------------------------------------------------------|

4467

```
template<address_mode address_mode = address_clamp>
const gather_return_type gather_red(const coordinates_type& coord) const restrict(amp);
```

Gathering the red component using predefined sampler objects based on specified address\_mode template argument.

#### Template parameters

|              |                                                                                      |
|--------------|--------------------------------------------------------------------------------------|
| address_mode | The address mode of the predefined sampler to use. static_assert with address_border |
|--------------|--------------------------------------------------------------------------------------|

#### Parameters

|       |                          |
|-------|--------------------------|
| coord | Coordinate for sampling. |
|-------|--------------------------|

|                     |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <b>Return value</b> | Rank 4 short vector containing the red component of the 4 texel values sampled. |
|---------------------|---------------------------------------------------------------------------------|

4468

```
template<address_mode address_mode = address_clamp>
const gather_return_type gather_green(const coordinates_type& coord) const restrict(amp);
```

Gathering the green component using predefined sampler objects based on specified address\_mode template argument.

#### Template parameters

|              |                                                                                      |
|--------------|--------------------------------------------------------------------------------------|
| address_mode | The address mode of the predefined sampler to use. static_assert with address_border |
|--------------|--------------------------------------------------------------------------------------|

#### Parameters

|       |                          |
|-------|--------------------------|
| coord | Coordinate for sampling. |
|-------|--------------------------|

|                     |                                                                                   |
|---------------------|-----------------------------------------------------------------------------------|
| <b>Return value</b> | Rank 4 short vector containing the green component of the 4 texel values sampled. |
|---------------------|-----------------------------------------------------------------------------------|

4469

```
template<address_mode address_mode = address_clamp>
const gather_return_type gather_blue(const coordinates_type& coord) const restrict(amp);
```

Gathering the blue component using predefined sampler objects based on specified address\_mode template argument.

**Template parameters**

|              |                                                                                      |
|--------------|--------------------------------------------------------------------------------------|
| address_mode | The address mode of the predefined sampler to use. static_assert with address_border |
|--------------|--------------------------------------------------------------------------------------|

**Parameters**

|       |                          |
|-------|--------------------------|
| coord | Coordinate for sampling. |
|-------|--------------------------|

|                     |                                                                                  |
|---------------------|----------------------------------------------------------------------------------|
| <b>Return value</b> | Rank 4 short vector containing the blue component of the 4 texel values sampled. |
|---------------------|----------------------------------------------------------------------------------|

4470

```
template<address_mode address_mode = address_clamp>
const gather_return_type gather_alpha(const coordinates_type& coord) const restrict(amp);
```

Gathering the alpha component using predefined sampler objects based on specified address\_mode template argument.

**Template parameters**

|              |                                                                                      |
|--------------|--------------------------------------------------------------------------------------|
| address_mode | The address mode of the predefined sampler to use. static_assert with address_border |
|--------------|--------------------------------------------------------------------------------------|

**Parameters**

|       |                          |
|-------|--------------------------|
| coord | Coordinate for sampling. |
|-------|--------------------------|

|                     |                                                                                   |
|---------------------|-----------------------------------------------------------------------------------|
| <b>Return value</b> | Rank 4 short vector containing the alpha component of the 4 texel values sampled. |
|---------------------|-----------------------------------------------------------------------------------|

4471

## 10.6 Global texture copy functions

C++ AMP provides a set of global copy functions which covers all the data transfer requirements between texture, texture\_view and iterators. These copy APIs also supports copying in and out sections of data from texture and texture\_view. Texture copy functions supports following source/destination pairs:

4472     • Copy from iterator to texture/texture\_view and vice-versa

4473     • Copy from texture/texture\_view to another texture/texture\_view and vice-versa

4478 The following functions do not participate in overload resolution unless template parameters *src\_type* and *dst\_type* are either *texture* or *texture\_view* types.

4479

4480

```
template <typename src_type>
void copy(const src_type& texture, void * dst, unsigned int dst_byte_size);
```

Copies raw texture data to a host-side buffer. The buffer must be laid out in accordance with the texture format and dimensions.

**Parameters**

|         |                                |
|---------|--------------------------------|
| texture | Source texture or texture_view |
|---------|--------------------------------|

|     |                                           |
|-----|-------------------------------------------|
| dst | Pointer to destination buffer on the host |
|-----|-------------------------------------------|

|               |                                           |
|---------------|-------------------------------------------|
| dst_byte_size | Number of bytes in the destination buffer |
|---------------|-------------------------------------------|

**Error condition**     **Exception thrown**

|                   |  |
|-------------------|--|
| Out of memory (*) |  |
|-------------------|--|

|                  |  |
|------------------|--|
| Buffer too small |  |
|------------------|--|

4481

4482     (\*) Out of memory errors may occur due to the need to allocate temporary buffers in some memory transfer scenarios.

4483

```
template <typename src_type>
void copy(const src_type& texture, const index<src_type::rank>& src_offset, const
extent<src_type::rank>& copy_extent, void * dst, unsigned int dst_byte_size);
```

Copies a section of a texture to a host-side buffer. The buffer must be laid out in accordance with the texture format and dimensions.

**Parameters**

|                        |                                           |
|------------------------|-------------------------------------------|
| texture                | Source texture or texture_view            |
| src_offset             | Offset into texture to begin copying from |
| copy_extent            | Extent of the section to copy             |
| dst                    | Pointer to destination buffer on the host |
| dst_byte_size          | Number of bytes in the destination buffer |
| <b>Error condition</b> | <b>Exception thrown</b>                   |
| Out of memory (*)      |                                           |
| Buffer too small       |                                           |

4484

```
template <typename dst_type>
void copy(const void * src, unsigned int src_byte_size, dst_type& texture);
```

Copies raw texture data to a device-side texture. The buffer must be laid out in accordance with the texture format and dimensions.

#### Parameters

|                        |                                      |
|------------------------|--------------------------------------|
| texture                | Destination texture or texture_view  |
| src                    | Pointer to source buffer on the host |
| src_byte_size          | Number of bytes in the source buffer |
| <b>Error condition</b> | <b>Exception thrown</b>              |
| Out of memory          |                                      |
| Buffer too small       |                                      |

4485

```
template <typename dst_type>
void copy(const void * src, unsigned int src_byte_size, dst_type& texture, const
index<dst_type::rank>& dst_offset, const extent<dst_type::rank>& copy_extent);
```

Copies raw texture data to a section of a texture. The buffer must be laid out in accordance with the texture format and dimensions.

#### Parameters

|                        |                                         |
|------------------------|-----------------------------------------|
| src                    | Pointer to source buffer on the host    |
| src_byte_size          | Number of bytes in the source buffer    |
| texture                | Destination texture or texture_view     |
| dst_offset             | Offset into texture to begin copying to |
| copy_extent            | Extent of the section to copy           |
| <b>Error condition</b> | <b>Exception thrown</b>                 |
| Out of memory          |                                         |
| Buffer too small       |                                         |

4486

```
template <typename InputIterator, typename dst_type>
void copy(InputIterator first, InputIterator last, dst_type& texture);
```

Copies raw texture data from a pair of iterators to a device-side texture. The iterated data must be laid out in accordance with the texture format and dimensions. The texture must not use different bits per scalar element than is the natural size of its element type.

#### Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| first           | First iterator                      |
| last            | End iterator                        |
| texture         | Destination texture or texture view |
| Error condition | Exception thrown                    |

|                  |  |
|------------------|--|
| Out of memory    |  |
| Buffer too small |  |

4487

4488

```
template <typename InputIterator, typename dst_type>
void copy(InputIterator first, InputIterator last, dst_type& texture, const
index<dst_type::rank>& dst_offset, const extent<dst_type::rank>& copy_extent);
```

Copies raw texture data from a pair of iterators to a section of a texture. The iterated data must be laid out in accordance with the texture format and dimensions. The texture must not use different bits per scalar element than is the natural size of its element type.

#### Parameters

|             |                                         |
|-------------|-----------------------------------------|
| first       | First iterator                          |
| last        | End iterator                            |
| texture     | Destination texture or texture view     |
| dst_offset  | Offset into texture to begin copying to |
| copy_extent | Extent of the section to copy           |

#### Error condition

#### Exception thrown

4489

```
template <typename src_type, typename OutputIterator>
void copy(const src_type& texture, OutputIterator dst);
```

Copies data from a texture to an output iterator. The iterated data must be laid out in accordance with the texture format and dimensions. The texture must not use different bits per scalar element than is the natural size of its element type.

#### Parameters

|         |                                |
|---------|--------------------------------|
| texture | Source texture or texture view |
| dst     | Destination iterator           |

#### Error condition

#### Exception thrown

4490

```
template <typename src_type, typename OutputIterator>
void copy(const src_type& texture, const index<src_type::rank>& src_offset, const
extent<src_type::rank>& copy_extent, OutputIterator dst);
```

Copies data from a section of a texture to an output iterator. The iterated data must be laid out in accordance with the texture format and dimensions. The texture must not use different bits per scalar element than is the natural size of its element type.

#### Parameters

|             |                                           |
|-------------|-------------------------------------------|
| texture     | Destination texture or texture view       |
| src_offset  | Offset into texture to begin copying from |
| copy_extent | Extent of the section to copy             |
| dst         | Destination iterator                      |

#### Error condition

#### Exception thrown

4491

```
template <typename src_type, typename dst_type>
```

```
void copy(const src_type& src_texture, dst_type& dst_texture);
```

Copies data between textures. Textures must have the same rank, dimension, bits per scalar element and number of mipmap levels.

#### Parameters

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| src_texture                       | Source texture or texture view      |
| dst_texture                       | Destination texture or texture_view |
| <b>Error condition</b>            | <b>Exception thrown</b>             |
| Out of memory                     |                                     |
| Incompatible dimensions           | concurrency::runtime_exception      |
| Different bits per scalar element | concurrency::runtime_exception      |
| Different number of mipmap levels | concurrency::runtime_exception      |

4492

4493

```
template <typename src_type, typename dst_type>
void copy(const src_type& src_texture, const index<src_type::rank>& src_offset, dst_type
dst_texture, const index<dst_type::rank>& dst_offset, const extent<dst_type::rank>&
copy_extent);
```

Copies data from a section of a texture to a section of another texture. Textures must have the same rank and must be distinct objects. Both the source and destination texture / texture\_view cannot have multiple mipmap level.

#### Parameters

|                                   |                                               |
|-----------------------------------|-----------------------------------------------|
| src_Texture                       | Source texture or texture view                |
| src_offset                        | Offset into src_Texture to begin copying from |
| dst_texture                       | Destination texture or texture_view           |
| dst_offset                        | Offset into dst to begin copying to           |
| copy_extent                       | Extent of the section to copy                 |
| <b>Error condition</b>            | <b>Exception thrown</b>                       |
| Out of memory                     |                                               |
| Source and destination identical  | concurrency::runtime_exception                |
| Different bits per scalar element | concurrency::runtime_exception                |
| Multiple mipmap levels            | concurrency::runtime_exception                |

4494

#### 10.6.1 Global async texture copy functions

For each `copy` function specified above, a `copy_async` function will also be provided, returning a `concurrency::completion_future`.

### 10.7 norm and unorm

The `norm` type is a single-precision floating point value that is normalized to the range [-1.0f, 1.0f]. The `unorm` type is a single-precision floating point value that is normalized to the range [0.0f, 1.0f].

#### 10.7.1 Synopsis

4502

4503

4504

4505

4506

```
class norm
{
public:
    norm() restrict(cpu, amp);
```

```

4507     explicit norm(float v) restrict(cpu, amp);
4508     explicit norm(unsigned int v) restrict(cpu, amp);
4509     explicit norm(int v) restrict(cpu, amp);
4510     explicit norm(double v) restrict(cpu, amp);
4511     norm(const norm& other) restrict(cpu, amp);
4512     norm(const unorm& other) restrict(cpu, amp);
4513
4514     norm& operator=(const norm& other) restrict(cpu, amp);
4515
4516     operator float(void) const restrict(cpu, amp);
4517
4518     norm& operator+=(const norm& other) restrict(cpu, amp);
4519     norm& operator-=(const norm& other) restrict(cpu, amp);
4520     norm& operator*=(const norm& other) restrict(cpu, amp);
4521     norm& operator/=(const norm& other) restrict(cpu, amp);
4522     norm& operator++() restrict(cpu, amp);
4523     norm operator++(int) restrict(cpu, amp);
4524     norm& operator--() restrict(cpu, amp);
4525     norm operator--(int) restrict(cpu, amp);
4526     norm operator-() restrict(cpu, amp);
4527 };
4528
4529 class unorm
4530 {
4531 public:
4532     unorm() restrict(cpu, amp);
4533     explicit unorm(float v) restrict(cpu, amp);
4534     explicit unorm(unsigned int v) restrict(cpu, amp);
4535     explicit unorm(int v) restrict(cpu, amp);
4536     explicit unorm(double v) restrict(cpu, amp);
4537     unorm(const unorm& other) restrict(cpu, amp);
4538     explicit unorm(const norm& other) restrict(cpu, amp);
4539
4540     unorm& operator=(const unorm& other) restrict(cpu, amp);
4541
4542     operator float() const restrict(cpu, amp);
4543
4544     unorm& operator+=(const unorm& other) restrict(cpu, amp);
4545     unorm& operator-=(const unorm& other) restrict(cpu, amp);
4546     unorm& operator*=(const unorm& other) restrict(cpu, amp);
4547     unorm& operator/=(const unorm& other) restrict(cpu, amp);
4548     unorm& operator++() restrict(cpu, amp);
4549     unorm operator++(int) restrict(cpu, amp);
4550     unorm& operator--() restrict(cpu, amp);
4551     unorm operator--(int) restrict(cpu, amp);
4552 };
4553
4554 unorm operator+(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4555 norm operator+(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4556
4557 unorm operator-(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4558 norm operator-(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4559
4560 unorm operator*(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4561 norm operator*(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4562
4563 unorm operator/(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4564 norm operator/(const norm& lhs, const norm& rhs) restrict(cpu, amp);

```

```

4565
4566     bool operator==(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4567     bool operator==(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4568
4569     bool operator!=(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4570     bool operator!=(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4571
4572     bool operator>(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4573     bool operator>(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4574
4575     bool operator<(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4576     bool operator<(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4577
4578     bool operator>=(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4579     bool operator>=(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4580
4581     bool operator<=(const unorm& lhs, const unorm& rhs) restrict(cpu, amp);
4582     bool operator<=(const norm& lhs, const norm& rhs) restrict(cpu, amp);
4583
4584 #define UNORM_MIN ((unorm)0.0f)
4585 #define UNORM_MAX ((unorm)1.0f)
4586 #define UNORM_ZERO ((norm)0.0f)
4587 #define NORM_ZERO ((norm)0.0f)
4588 #define NORM_MIN ((norm)-1.0f)
4589 #define NORM_MAX ((norm)1.0f)
4590

```

### 4591 10.7.2 Constructors and Assignment

4592 An object of type *norm* or *unorm* can be explicitly constructed from one of the following types:

- 4593 • *float*
- 4594 • *double*
- 4595 • *int*
- 4596 • *unsigned int*
- 4597 • *norm*
- 4598 • *unorm*

4599 In all these constructors, the object is initialized by first converting the argument to the *float* data type, and then clamping  
4600 the value into the range defined by the type.

4601 4602 Assignment from *norm* to *norm* is defined, as is assignment from *unorm* to *unorm*. Assignment from other types requires  
4603 an explicit conversion.

### 4604 10.7.3 Operators

4605 All arithmetic operators that are defined for the *float* type are defined for *norm* and *unorm* as well. For each supported  
4606 operator  $\oplus$ , the result is computed in single-precision floating point arithmetic, and if required is then clamped back to the  
4607 appropriate range.

4608 4609 Both *norm* and *unorm* are implicitly convertible to *float*.

## 4610 10.8 Short Vector Types

4611 C++ AMP defines a set of short vector types (of length 2, 3, and 4) which are based on one of the following scalar types: {*int*,  
4612 *unsigned int*, *float*, *double*, *norm*, *unorm*}, and are named as summarized in the following table:  
4613

| Scalar Type | Length |   |   |
|-------------|--------|---|---|
|             | 2      | 3 | 4 |
|             |        |   |   |

|                     |                   |                   |                   |
|---------------------|-------------------|-------------------|-------------------|
| <b>int</b>          | int_2, int2       | int_3, int3       | int_4, int4       |
| <b>unsigned int</b> | uint_2, uint2     | uint_3, uint3     | uint_4, uint4     |
| <b>float</b>        | float_2, float2   | float_3, float3   | float_4, float4   |
| <b>double</b>       | double_2, double2 | double_3, double3 | double_4, double4 |
| <b>norm</b>         | norm_2, norm2     | norm_3, norm3     | norm_4, norm4     |
| <b>unorm</b>        | unorm_2, unorm2   | unorm_3, unorm3   | unorm_4, unorm4   |

4614

4615 There is no functional difference between the type `scalar_N` and `scalarN`. `scalarN` type is available in the `graphics::direct3d` namespace.

4617

4618 Unlike `index<N>` and `extent<N>`, short vector types have no notion of significance or endian-ness, as they are not assumed  
4619 to be describing the shape of data or compute (even though a user might choose to use them this way). Also unlike extents  
4620 and indices, short vector types cannot be indexed using the subscript operator.

4621

4622 Components of short vector types can be accessed by name. By convention, short vector type components can use either  
4623 Cartesian coordinate names ("x", "y", "z", and "w"), or color scalar element names ("r", "g", "b", and "a").

4624

- For length-2 vectors, only the names "x", "y" and "r", "g" are available.
- For length-3 vectors, only the names "x", "y", "z", and "r", "g", "b" are available.
- For length-4 vectors, the full set of names "x", "y", "z", "w", and "r", "g", "b", "a" are available.

4627

4628 Note that the names derived from the color channel space (rgba) are available only as properties, not as getter and setter  
functions.

### 4629 10.8.1 Synopsis

4630

4631 Because the full synopsis of all the short vector types is quite large, this section will summarize the basic structure of all the  
4632 short vector types.

4633

4634 In the summary class definition below the word "scalartype" is one of { `int`, `uint`, `float`, `double`, `norm`, `unorm` }. The value `N` is  
4635 2, 3 or 4.

4636

```
4637 class scalartype_N
4638 {
4639 public:
4640     typedef scalartype value_type;
4641     static const int size = N;
4642
4643     scalartype_N() restrict(cpu, amp);
4644     scalartype_N(scalartype value) restrict(cpu, amp);
4645     scalartype_N(const scalartype_N& other) restrict(cpu, amp);
4646
4647     // Component-wise constructor... see 10.8.2.1 Constructors from components
4648
4649     // Constructors that explicitly convert from other short vector types...
4650     // See 10.8.2.2 Explicit conversion constructors.
4651
4652     scalartype_N& operator=(const scalartype_N& other) restrict(cpu, amp);
4653
4654     // Operators
4655     scalartype_N& operator++() restrict(cpu, amp);
4656     scalartype_N operator++(int) restrict(cpu, amp);
4657     scalartype_N& operator--() restrict(cpu, amp);
```

```

4658     scalartype_N operator--(int) restrict(cpu, amp);
4659     scalartype_N& operator+=(const scalartype_N& rhs) restrict(cpu, amp);
4660     scalartype_N& operator-=(const scalartype_N& rhs) restrict(cpu, amp);
4661     scalartype_N& operator*=(const scalartype_N& rhs) restrict(cpu, amp);
4662     scalartype_N& operator/=(const scalartype_N& rhs) restrict(cpu, amp);
4663
4664     // Unary negation: not for scalartype == uint or unorm
4665     scalartype_N operator-() const restrict(cpu, amp);
4666
4667     // More integer operators (only for scalartype == int or uint)
4668     scalartype_N operator~() const restrict(cpu, amp);
4669     scalartype_N& operator%=(const scalartype_N& rhs) restrict(cpu, amp);
4670     scalartype_N& operator^=(const scalartype_N& rhs) restrict(cpu, amp);
4671     scalartype_N& operator|=(const scalartype_N& rhs) restrict(cpu, amp);
4672     scalartype_N& operator&=(const scalartype_N& rhs) restrict(cpu, amp);
4673     scalartype_N& operator>>=(const scalartype_N& rhs) restrict(cpu, amp);
4674     scalartype_N& operator<<=(const scalartype_N& rhs) restrict(cpu, amp);
4675
4676     // Component accessors and properties (a.k.a. swizzling):
4677     // See 10.8.3 Component Access (Swizzling)
4678 };
4679
4680     scalartype_N operator+(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4681     scalartype_N operator-(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4682     scalartype_N operator*(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4683     scalartype_N operator/(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4684     bool operator==(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4685     bool operator!=(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4686
4687     // More integer operators (only for scalartype == int or uint)
4688     scalartype_N operator%(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4689     scalartype_N operator^(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4690     scalartype_N operator|(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4691     scalartype_N operator&(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4692     scalartype_N operator<<=(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);
4693     scalartype_N operator>>=(const scalartype_N& lhs, const scalartype_N& rhs) restrict(cpu, amp);

```

#### 4694 10.8.2 Constructors

|                                                 |
|-------------------------------------------------|
| <code>scalartype_N() restrict(cpu, amp);</code> |
|-------------------------------------------------|

4695 Default constructor. Initializes all components to zero.

4696

|                                                                 |
|-----------------------------------------------------------------|
| <code>scalartype_N(scalartype value) restrict(cpu, amp);</code> |
|-----------------------------------------------------------------|

4697 Initializes all components of the short vector to 'value'.

**Parameters:**

|                    |                                                                   |
|--------------------|-------------------------------------------------------------------|
| <code>value</code> | The value with which to initialize each component of this vector. |
|--------------------|-------------------------------------------------------------------|

4698

|                                                                              |
|------------------------------------------------------------------------------|
| <code>scalartype_N(const scalartype_N&amp; other) restrict(cpu, amp);</code> |
|------------------------------------------------------------------------------|

Copy constructor. Copies the contents of 'other' to 'this'.

**Parameters:**

|                    |                                 |
|--------------------|---------------------------------|
| <code>other</code> | The source vector to copy from. |
|--------------------|---------------------------------|

## 4699 10.8.2.1 Constructors from components

4700 A short vector type can also be constructed with values for each of its components.

4701

```
scalartype_2(scalartype v1, scalartype v2) restrict(cpu,amp); // only for length 2
scalartype_3(scalartype v1, scalartype v2, scalartype v3) restrict(cpu,amp); // only for length 3
scalartype_4(scalartype v1, scalartype v2,
            scalartype v3, scalartype v4) restrict(cpu,amp); // only for length 4
```

Creates a short vector with the provided initialize values for each component.

**Parameters:**

|    |                                                                |
|----|----------------------------------------------------------------|
| v1 | The value with which to initialize the "x" (or "r") component. |
| v2 | The value with which to initialize the "y" (or "g") component  |
| v3 | The value with which to initialize the "z" (or "b") component. |
| v4 | The value with which to initialize the "w" (or "a") component  |

4702

## 4703 10.8.2.2 Explicit conversion constructors

4704 A short vector of type *scalartype1\_N* can be constructed from an object of type *scalartype2\_N*, as long as *N* is the same in  
4705 both types. For example, a *uint\_4* can be constructed from a *float\_4*.

4706

```
explicit scalartype_N(const int_N& other) restrict(cpu,amp);
explicit scalartype_N(const uint_N& other) restrict(cpu,amp);
explicit scalartype_N(const float_N& other) restrict(cpu,amp);
explicit scalartype_N(const double_N& other) restrict(cpu,amp);
explicit scalartype_N(const norm_N& other) restrict(cpu,amp);
explicit scalartype_N(const unorm_N& other) restrict(cpu,amp);
```

Construct a short vector from a differently-typed short vector, performing an explicit conversion. Note that in the above list of 6 constructors, each short vector type will have 5 of these.

**Parameters:**

|       |                                         |
|-------|-----------------------------------------|
| other | The source vector to copy/convert from. |
|-------|-----------------------------------------|

## 4707 10.8.3 Component Access (Swizzling)

4708 The components of a short vector may be accessed in a large variety of ways, depending on the length of the short vector.

4709

- As single scalar components ( $N \geq 2$ )
- As reference to single scalar components ( $N \geq 2$ )
- As pairs of components, in any permutation ( $N \geq 2$ )
- As triplets of components, in any permutation ( $N \geq 3$ )
- As quadruplets of components, in any permutation ( $N = 4$ ).

4710

4711 Because the permutations of such component accessors are so large, they are described here using symmetric group  
4712 notation. In such notation,  $S_{xy}$  represents all permutations of the letters  $x$  and  $y$ , namely  $xy$  and  $yx$ . Similarly,  $S_{xyz}$  represents  
4713 all  $3! = 6$  permutations of the letters  $x$ ,  $y$ , and  $z$ , namely  $xy$ ,  $xz$ ,  $yx$ ,  $yz$ ,  $zx$ , and  $zy$ .

4714

4715 Recall that the  $z$  (or  $b$ ) component of a short vector is only available for vector lengths 3 and 4. The  $w$  (or  $a$ ) component of a  
4716 short vector is only available for vector length 4.

4717

4718

4719

4720

4721

4722

### 10.8.3.1 Single-component access

```
scalartype get_x() const restrict(cpu, amp);
scalartype get_y() const restrict(cpu, amp);
scalartype get_z() const restrict(cpu, amp);
scalartype get_w() const restrict(cpu, amp);

void set_x(scalartype v) const restrict(cpu, amp);
void set_y(scalartype v) const restrict(cpu, amp);
void set_z(scalartype v) const restrict(cpu, amp);
void set_w(scalartype v) const restrict(cpu, amp);

__declspec(property (get=get_x, put=set_x)) scalartype x;
__declspec(property (get=get_y, put=set_y)) scalartype y;
__declspec(property (get=get_z, put=set_z)) scalartype z;
__declspec(property (get=get_w, put=set_w)) scalartype w;
__declspec(property (get=get_x, put=set_x)) scalartype r;
__declspec(property (get=get_y, put=set_y)) scalartype g;
__declspec(property (get=get_z, put=set_z)) scalartype b;
__declspec(property (get=get_w, put=set_w)) scalartype a;
```

These functions (and properties) allow access to individual components of a short vector type. Note that the properties in the "rgba" space map to functions in the "xyzw" space.

4723

4724

### 10.8.3.2 Reference to single-component access

```
scalartype& ref_x() restrict(cpu, amp);
scalartype& ref_y() restrict(cpu, amp);
scalartype& ref_z() restrict(cpu, amp);
scalartype& ref_w() restrict(cpu, amp);

scalartype& ref_r() restrict(cpu, amp);
scalartype& ref_g() restrict(cpu, amp);
scalartype& ref_b() restrict(cpu, amp);
scalartype& ref_a() restrict(cpu, amp);
```

These functions return references to individual components of a short vector type. They can be used to perform atomic operations on individual components of a short vector.

4725

4726

4727

### 10.8.3.3 Two-component access

```
scalartype_2 get_Sxy() const restrict(cpu, amp);
scalartype_2 get_Sxz() const restrict(cpu, amp);
scalartype_2 get_Sxw() const restrict(cpu, amp);
scalartype_2 get_Syz() const restrict(cpu, amp);
scalartype_2 get_Syw() const restrict(cpu, amp);
scalartype_2 get_Szw() const restrict(cpu, amp);

void set_Sxy(scalartype_2 v) restrict(cpu, amp);
void set_Sxz(scalartype_2 v) restrict(cpu, amp);
void set_Sxw(scalartype_2 v) restrict(cpu, amp);
void set_Syz(scalartype_2 v) restrict(cpu, amp);
void set_Syw(scalartype_2 v) restrict(cpu, amp);
void set_Szw(scalartype_2 v) restrict(cpu, amp);

__declspec(property (get=get_Sxy, put=set_Sxy)) scalartype_2 Sxy;
__declspec(property (get=get_Sxz, put=set_Sxz)) scalartype_2 Sxz;
```

```
_declspec(property (get=get_Sxw, put=set_Sxw)) scalartype_2 Sxw;
_declspec(property (get=get_Syz, put=set_Syz)) scalartype_2 Syz;
_declspec(property (get=get_Syw, put=set_Syw)) scalartype_2 Syw;
_declspec(property (get=get_Szw, put=set_Szw)) scalartype_2 Szw;
_declspec(property (get=get_Sxy, put=set_Sxy)) scalartype_2 Srg;
_declspec(property (get=get_Sxz, put=set_Sxz)) scalartype_2 Srb;
_declspec(property (get=get_Sxw, put=set_Sxw)) scalartype_2 Sra;
_declspec(property (get=get_Syz, put=set_Syz)) scalartype_2 Sgb;
_declspec(property (get=get_Syw, put=set_Syw)) scalartype_2 Sga;
_declspec(property (get=get_Szw, put=set_Szw)) scalartype_2 Sbba;
```

These functions (and properties) allow access to pairs of components. For example:

```
int_3 f3(1,2,3);
int_2 yz = f3.yz; // yz = (2,3)
```

4728

4729

#### 10.8.3.4 Three-component access

```
scalartype_3 get_Sxyz() const restrict(cpu, amp);
scalartype_3 get_Sxyw() const restrict(cpu, amp);
scalartype_3 get_Sxzw() const restrict(cpu, amp);
scalartype_3 get_Syzw() const restrict(cpu, amp);

void set_Sxyz(scalartype_3 v) restrict(cpu, amp);
void set_Sxyw(scalartype_3 v) restrict(cpu, amp);
void set_Sxzw(scalartype_3 v) restrict(cpu, amp);
void set_Syzw(scalartype_3 v) restrict(cpu, amp);

_declspec(property (get=get_Sxyz, put=set_Sxyz)) scalartype_3 Sxyz;
_declspec(property (get=get_Sxyw, put=set_Sxyw)) scalartype_3 Sxyw;
_declspec(property (get=get_Sxzw, put=set_Sxzw)) scalartype_3 Sxzw;
_declspec(property (get=get_Syzw, put=set_Syzw)) scalartype_3 Syzw;
_declspec(property (get=get_Sxyz, put=set_Sxyz)) scalartype_3 Srg;
_declspec(property (get=get_Sxyw, put=set_Sxyw)) scalartype_3 Srga;
_declspec(property (get=get_Sxzw, put=set_Sxzw)) scalartype_3 Srba;
_declspec(property (get=get_Syzw, put=set_Syzw)) scalartype_3 Sgba;
```

These functions (and properties) allow access to triplets of components (for vectors of length 3 or 4). For example:

```
int_4 f3(1,2,3,4);
int_3 wzy = f3.wzy; // wzy = (4,3,2)
```

4730

4731

#### 10.8.3.5 Four-component access

```
scalartype_4 get_Sxyzw() const restrict(cpu, amp);

void set_Sxyzw(scalartype_4 v) restrict(cpu, amp);

_declspec(property (get=get_Sxyzw, put=set_Sxyzw)) scalartype_4 Sxyzw
_declspec(property (get=get_Sxyzw, put=set_Sxyzw)) scalartype_4 Srgba
```

These functions (and properties) allow access to all four components (obviously, only for vectors of length 4). For example:

```
int_4 f3(1,2,3,4);
int_4 wzyx = f3.wzyx; // wzyx = (4,3,2,1)
```

4732

## 10.9 Template Versions of Short Vector Types

The template class `short_vector` provides metaprogramming definitions of the above short vector types. These are useful for programming short vectors generically. In general, the type “`scalartype_N`” is equivalent to “`short_vector<scalartype,N>::type`”.

4737 **10.9.1 Synopsis**

```

4738 template<typename scalar_type, int size> struct short_vector
4739 {
4740     short_vector()
4741     {
4742         static_assert(false, "short_vector is not supported for this scalar type (T) and length
4743 (N)");
4744     }
4745 };
4746 };
4747
4748 template<>
4749 struct short_vector<unsigned int, 1>
4750 {
4751     typedef unsigned int type;
4752 };
4753
4754 template<>
4755 struct short_vector<unsigned int, 2>
4756 {
4757     typedef uint_2 type;
4758 };
4759
4760 template<>
4761 struct short_vector<unsigned int, 3>
4762 {
4763     typedef uint_3 type;
4764 };
4765
4766 template<>
4767 struct short_vector<unsigned int, 4>
4768 {
4769     typedef uint_4 type;
4770 };
4771
4772 template<>
4773 struct short_vector<int, 1>
4774 {
4775     typedef int type;
4776 };
4777
4778 template<>
4779 struct short_vector<int, 2>
4780 {
4781     typedef int_2 type;
4782 };
4783
4784 template<>
4785 struct short_vector<int, 3>
4786 {
4787     typedef int_3 type;
4788 };
4789
4790 template<>
4791 struct short_vector<int, 4>
4792 {
4793     typedef int_4 type;
4794 };

```

```
4795 template<>
4796 struct short_vector<float, 1>
4797 {
4798     typedef float type;
4799 };
4800
4801
4802 template<>
4803 struct short_vector<float, 2>
4804 {
4805     typedef float_2 type;
4806 };
4807
4808 template<>
4809 struct short_vector<float, 3>
4810 {
4811     typedef float_3 type;
4812 };
4813
4814 template<>
4815 struct short_vector<float, 4>
4816 {
4817     typedef float_4 type;
4818 };
4819
4820 template<>
4821 struct short_vector<unorm, 1>
4822 {
4823     typedef unorm type;
4824 };
4825
4826 template<>
4827 struct short_vector<unorm, 2>
4828 {
4829     typedef unorm_2 type;
4830 };
4831
4832 template<>
4833 struct short_vector<unorm, 3>
4834 {
4835     typedef unorm_3 type;
4836 };
4837
4838 template<>
4839 struct short_vector<unorm, 4>
4840 {
4841     typedef unorm_4 type;
4842 };
4843
4844 template<>
4845 struct short_vector<norm, 1>
4846 {
4847     typedef norm type;
4848 };
4849
4850 template<>
4851 struct short_vector<norm, 2>
4852 {
```

```

4853     typedef norm_2 type;
4854 };
4855
4856 template<>
4857 struct short_vector<norm, 3>
4858 {
4859     typedef norm_3 type;
4860 };
4861
4862 template<>
4863 struct short_vector<norm, 4>
4864 {
4865     typedef norm_4 type;
4866 };
4867
4868 template<>
4869 struct short_vector<double, 1>
4870 {
4871     typedef double type;
4872 };
4873
4874 template<>
4875 struct short_vector<double, 2>
4876 {
4877     typedef double_2 type;
4878 };
4879
4880 template<>
4881 struct short_vector<double, 3>
4882 {
4883     typedef double_3 type;
4884 };
4885
4886 template<>
4887 struct short_vector<double, 4>
4888 {
4889     typedef double_4 type;
4890 };
4891

```

#### 4892 10.9.2 short\_vector<T,N> type equivalences

4893 The equivalences of the template types “short\_vector<scalartype,N>::type” to “scalartype\_N” are listed in the table below:

4894

| short_vector template               | Equivalent type |
|-------------------------------------|-----------------|
| short_vector<unsigned int, 1>::type | unsigned int    |
| short_vector<unsigned int, 2>::type | uint_2          |
| short_vector<unsigned int, 3>::type | uint_3          |
| short_vector<unsigned int, 4>::type | uint_4          |
| short_vector<int, 1>::type          | int             |
| short_vector<int, 2>::type          | int_2           |
| short_vector<int, 3>::type          | int_3           |
| short_vector<int, 4>::type          | int_4           |
| short_vector<float, 1>::type        | float           |

|                               |          |
|-------------------------------|----------|
| short_vector<float, 2>::type  | float_2  |
| short_vector<float, 3>::type  | float_3  |
| short_vector<float, 4>::type  | float_4  |
| short_vector<unorm, 1>::type  | unorm    |
| short_vector<unorm, 2>::type  | unorm_2  |
| short_vector<unorm, 3>::type  | unorm_3  |
| short_vector<unorm, 4>::type  | unorm_4  |
| short_vector<norm, 1>::type   | norm     |
| short_vector<norm, 2>::type   | norm_2   |
| short_vector<norm, 3>::type   | norm_3   |
| short_vector<norm, 4>::type   | norm_4   |
| short_vector<double, 1>::type | double   |
| short_vector<double, 2>::type | double_2 |
| short_vector<double, 3>::type | double_3 |
| short_vector<double, 4>::type | double_4 |

4895

4896 **10.10 Template class short\_vector\_traits**4897 The template class short\_vector\_traits provides the ability to reflect on the supported short vector types and obtain the  
4898 length of the vector and the underlying scalar type.4899 **10.10.1 Synopsis**

```

4900
4901 template<typename type> struct short_vector_traits
4902 {
4903     short_vector_traits()
4904     {
4905         static_assert(false, "short_vector_traits is not supported for this type (type)");
4906     }
4907 };
4908
4909 template<>
4910 struct short_vector_traits<unsigned int>
4911 {
4912     typedef unsigned int value_type;
4913     static int const size = 1;
4914 };
4915
4916 template<>
4917 struct short_vector_traits<uint_2>
4918 {
4919     typedef unsigned int value_type;
4920     static int const size = 2;
4921 };
4922
4923 template<>
4924 struct short_vector_traits<uint_3>
4925 {
4926     typedef unsigned int value_type;
4927     static int const size = 3;

```

```
4928 };
4929 template<>
4930 struct short_vector_traits<uint_4>
4931 {
4932     typedef unsigned int value_type;
4933     static int const size = 4;
4934 };
4935
4936 template<>
4937 struct short_vector_traits<int>
4938 {
4939     typedef int value_type;
4940     static int const size = 1;
4941 };
4942
4943 template<>
4944 struct short_vector_traits<int_2>
4945 {
4946     typedef int value_type;
4947     static int const size = 2;
4948 };
4949
4950 template<>
4951 struct short_vector_traits<int_3>
4952 {
4953     typedef int value_type;
4954     static int const size = 3;
4955 };
4956
4957 template<>
4958 struct short_vector_traits<int_4>
4959 {
4960     typedef int value_type;
4961     static int const size = 4;
4962 };
4963
4964 template<>
4965 struct short_vector_traits<float>
4966 {
4967     typedef float value_type;
4968     static int const size = 1;
4969 };
4970
4971 template<>
4972 struct short_vector_traits<float_2>
4973 {
4974     typedef float value_type;
4975     static int const size = 2;
4976 };
4977
4978 template<>
4979 struct short_vector_traits<float_3>
4980 {
4981     typedef float value_type;
4982     static int const size = 3;
4983 };
4984
4985
```

```
4986 template<>
4987 struct short_vector_traits<float_4>
4988 {
4989     typedef float value_type;
4990     static int const size = 4;
4991 };
4992
4993 template<>
4994 struct short_vector_traits<unorm>
4995 {
4996     typedef unorm value_type;
4997     static int const size = 1;
4998 };
4999
5000 template<>
5001 struct short_vector_traits<unorm_2>
5002 {
5003     typedef unorm value_type;
5004     static int const size = 2;
5005 };
5006
5007 template<>
5008 struct short_vector_traits<unorm_3>
5009 {
5010     typedef unorm value_type;
5011     static int const size = 3;
5012 };
5013
5014 template<>
5015 struct short_vector_traits<unorm_4>
5016 {
5017     typedef unorm value_type;
5018     static int const size = 4;
5019 };
5020
5021 template<>
5022 struct short_vector_traits<norm>
5023 {
5024     typedef norm value_type;
5025     static int const size = 1;
5026 };
5027
5028 template<>
5029 struct short_vector_traits<norm_2>
5030 {
5031     typedef norm value_type;
5032     static int const size = 2;
5033 };
5034
5035 template<>
5036 struct short_vector_traits<norm_3>
5037 {
5038     typedef norm value_type;
5039     static int const size = 3;
5040 };
5041
5042 template<>
5043 struct short_vector_traits<norm_4>
```

```

5044     {
5045         typedef norm value_type;
5046         static int const size = 4;
5047     };
5048
5049     template<>
5050     struct short_vector_traits<double>
5051     {
5052         typedef double value_type;
5053         static int const size = 1;
5054     };
5055
5056     template<>
5057     struct short_vector_traits<double_2>
5058     {
5059         typedef double value_type;
5060         static int const size = 2;
5061     };
5062
5063     template<>
5064     struct short_vector_traits<double_3>
5065     {
5066         typedef double value_type;
5067         static int const size = 3;
5068     };
5069
5070     template<>
5071     struct short_vector_traits<double_4>
5072     {
5073         typedef double value_type;
5074         static int const size = 4;
5075     };

```

## 10.10.2 Typedefs

5076

**typedef scalar\_type value\_type;**

Introduces a **typedef** identifying the underling scalar type of the vector type. **scalar\_type** depends on the instantiation of class **short\_vector\_types** used. This is summarized in the list below

| Instantiated Type                              | Scalar Type  |
|------------------------------------------------|--------------|
| <b>short_vector_traits&lt;unsigned int&gt;</b> | unsigned int |
| <b>short_vector_traits&lt;uint_2&gt;</b>       | unsigned int |
| <b>short_vector_traits&lt;uint_3&gt;</b>       | unsigned int |
| <b>short_vector_traits&lt;uint_4&gt;</b>       | unsigned int |
| <b>short_vector_traits&lt;int&gt;</b>          | int          |
| <b>short_vector_traits&lt;int_2&gt;</b>        | int          |
| <b>short_vector_traits&lt;int_3&gt;</b>        | int          |
| <b>short_vector_traits&lt;int_4&gt;</b>        | int          |
| <b>short_vector_traits&lt;float&gt;</b>        | float        |
| <b>short_vector_traits&lt;float_2&gt;</b>      | float        |
| <b>short_vector_traits&lt;float_3&gt;</b>      | float        |
| <b>short_vector_traits&lt;float_4&gt;</b>      | float        |
| <b>short_vector_traits&lt;unorm&gt;</b>        | unorm        |

|                               |        |
|-------------------------------|--------|
| short_vector_traits<unorm_2>  | unorm  |
| short_vector_traits<unorm_3>  | unorm  |
| short_vector_traits<unorm_4>  | unorm  |
| short_vector_traits<norm>     | norm   |
| short_vector_traits<norm_2>   | norm   |
| short_vector_traits<norm_3>   | norm   |
| short_vector_traits<norm_4>   | norm   |
| short_vector_traits<double>   | double |
| short_vector_traits<double_2> | double |
| short_vector_traits<double_3> | double |
| short_vector_traits<double_4> | double |

5078

### 10.10.3 Members

5079

`static int const size;`

Introduces a static constant integer specifying the number of elements in the short vector type, based on the table below:

| Instantiated Type                 | Size |
|-----------------------------------|------|
| short_vector_traits<unsigned int> | 1    |
| short_vector_traits<uint_2>       | 2    |
| short_vector_traits<uint_3>       | 3    |
| short_vector_traits<uint_4>       | 4    |
| short_vector_traits<int>          | 1    |
| short_vector_traits<int_2>        | 2    |
| short_vector_traits<int_3>        | 3    |
| short_vector_traits<int_4>        | 4    |
| short_vector_traits<float>        | 1    |
| short_vector_traits<float_2>      | 2    |
| short_vector_traits<float_3>      | 3    |
| short_vector_traits<float_4>      | 4    |
| short_vector_traits<unorm>        | 1    |
| short_vector_traits<unorm_2>      | 2    |
| short_vector_traits<unorm_3>      | 3    |
| short_vector_traits<unorm_4>      | 4    |
| short_vector_traits<norm>         | 1    |
| short_vector_traits<norm_2>       | 2    |
| short_vector_traits<norm_3>       | 3    |
| short_vector_traits<norm_4>       | 4    |
| short_vector_traits<double>       | 1    |
| short_vector_traits<double_2>     | 2    |
| short_vector_traits<double_3>     | 3    |
| short_vector_traits<double_4>     | 4    |

5080

## 5081 11 D3D interoperability (Optional)

5082  
 5083 The C++ AMP runtime provides functions for D3D interoperability, enabling seamless use of D3D resources for compute in  
 5084 C++ AMP code as well as allow use of resources created in C++ AMP in D3D code, without the creation of redundant  
 5085 intermediate copies. These features allow users to incrementally accelerate the compute intensive portions of their DirectX  
 5086 applications using C++ AMP and use the D3D API on data produced from C++ AMP computations.

5087  
 5088 The following D3D interoperability functions and classes are available in the [direct3d](#) namespace:  
 5089

```
accelerator_view create_accelerator_view(IUnknown* D3d_device, queuing_mode qmode =
queuing_mode_automatic)
```

Creates a new [accelerator\\_view](#) from an existing Direct3D device interface pointer. On failure the function throws a [runtime\\_exception](#) exception. On success, the reference count of the parameter is incremented by making a [AddRef](#) call on the interface to record the C++ AMP reference to the interface, and users can safely [Release](#) the object when no longer required in their DirectX code.

The [accelerator\\_view](#) created using this function is thread-safe just as any C++ AMP created [accelerator\\_view](#), allowing concurrent submission of commands to it from multiple host threads. However, concurrent use of the [accelerator\\_view](#) and the raw [ID3D11Device](#) interface from multiple host threads must be properly synchronized by users to ensure mutual exclusion. Unsynchronized concurrent usage of the [accelerator\\_view](#) and the raw [ID3D11Device](#) interface will result in undefined behavior.

The C++ AMP runtime provides detailed error information in debug mode using the Direct3D Debug layer. However, if the Direct3D device passed to the above function was not created with the [D3D11\\_CREATE\\_DEVICE\\_DEBUG](#) flag, the C++ AMP debug mode detailed error information support will be unavailable.

### Parameters:

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>D3d_device</i>   | An AMP supported D3D device interface pointer to be used to create the accelerator_view. The parameter must meet all of the following conditions for successful creation of a accelerator_view: <ol style="list-style-type: none"> <li>1) Must be a supported D3D device interface. For this release, only ID3D11Device interface is supported.</li> <li>2) The device must have an AMP supported feature level. For this release this means a D3D_FEATURE_LEVEL_11_0. or D3D_FEATURE_LEVEL_11_1</li> <li>3) The D3D Device should not have been created with the "D3D11_CREATE_DEVICE_SINGLETHREADED" flag.</li> </ol> |
| <i>queuing_mode</i> | The queuing_mode to be used for the newly created accelerator_view. This parameter has a default value of queuing_mode_automatic.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

### Return Value:

The newly created accelerator\_view object.

### Exceptions:

|                                                   |                                |
|---------------------------------------------------|--------------------------------|
| Failed to create accelerator_view from D3D device | concurrency::runtime_exception |
| NULL D3D device pointer                           | concurrency::runtime_exception |

```
accelerator_view create_accelerator_view(accelerator& accl, bool disable_timeout, queuing_mode
qmode = queuing_mode_automatic)
```

Creates and returns a new [accelerator\\_view](#) on the specified accelerator. This method provides users control over whether

GPU timeout should be disabled for the newly created accelerator\_view, through the "disable\_timeout" boolean parameter. This corresponds to the D3D11\_CREATE\_DEVICE\_DISABLE\_GPU\_TIMEOUT flag for Direct3D device creation and is used to indicate if the operating system should allow workloads that take more than 2 seconds to execute, without resetting the device per the Windows timeout detection and recovery mechanism. Use of this flag is recommended if you need to perform time consuming tasks on the accelerator\_view.

**Parameters:**

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>acc1</i>            | The accelerator on which the new accelerator_view is to be created.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <i>disable_timeout</i> | A boolean parameter that specifies whether timeout should be disabled for the newly created accelerator_view. This corresponds to the D3D11_CREATE_DEVICE_DISABLE_GPU_TIMEOUT flag for Direct3D device creation and is used to indicate if the operating system should allow workloads that take more than 2 seconds to execute, without resetting the device per the Windows timeout detection and recovery mechanism. Use of this flag is recommended if you need to perform time consuming tasks on the accelerator_view. |
| <i>queuing_mode</i>    | The queuing_mode to be used for the newly created accelerator_view. This parameter has a default value of queuing_mode_automatic.                                                                                                                                                                                                                                                                                                                                                                                            |

**Return Value:**

The newly created accelerator\_view object.

**Exceptions:**

|                                   |                                |
|-----------------------------------|--------------------------------|
| Failed to create accelerator_view | concurrency::runtime_exception |
|-----------------------------------|--------------------------------|

5092

```
bool is_timeout_disabled(const accelerator_view& av);
```

Returns a boolean flag indicating if timeout is disabled for the specified accelerator\_view. This corresponds to the D3D11\_CREATE\_DEVICE\_DISABLE\_GPU\_TIMEOUT flag for Direct3D device creation.

**Parameters:**

|           |                                                                               |
|-----------|-------------------------------------------------------------------------------|
| <i>av</i> | The accelerator_view for which the timeout disabled setting is to be queried. |
|-----------|-------------------------------------------------------------------------------|

**Return Value:**

A boolean flag indicating if timeout is disabled for the specified accelerator\_view.

5093

```
IUnknown * get_device(const accelerator_view& av);
```

Returns a D3D device interface pointer underlying the passed accelerator\_view. Fails with a "runtime\_exception" exception if the passed accelerator\_view is not a D3D device accelerator view. On success, it increments the reference count of the D3D device interface by calling "AddRef" on the interface. Users must call "Release" on the returned interface after they are finished using it, for proper reclamation of the resources associated with the object.

Concurrent use of the accelerator\_view and the raw ID3D11Device interface from multiple host threads must be properly synchronized by users to ensure mutual exclusion. Unsynchronized concurrent usage of the accelerator\_view and the raw ID3D11Device interface will result in undefined behavior.

**Parameters:**

|           |                                                                           |
|-----------|---------------------------------------------------------------------------|
| <i>av</i> | The accelerator_view object for which the D3D device interface is needed. |
|-----------|---------------------------------------------------------------------------|

**Return Value:**

A IUnknown interface pointer corresponding to the D3D device underlying the passed accelerator\_view. Users must use the [QueryInterface](#) member function on the returned interface to obtain the correct D3D device interface pointer.

**Exceptions:**

|                                                       |                                |
|-------------------------------------------------------|--------------------------------|
| Cannot get D3D device from a non-D3D accelerator_view | concurrency::runtime_exception |
|-------------------------------------------------------|--------------------------------|

5094  
5095

```
template <typename T, int N>
array<T,N> make_array(const extent<N>& extent, const accelerator_view& av, IUnknown* D3d_buffer);
```

Creates an array with the specified extents on the specified accelerator\_view from an existing Direct3D buffer interface pointer. On failure the member function throws a *runtime\_exception* exception. On success, the reference count of the Direct3D buffer object is incremented by making an *AddRef* call on the interface to record the C++ AMP reference to the interface, and users can safely *Release* the object when no longer required in their DirectX code.

**Parameters:**

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| extent     | The extent of the array to be created.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| av         | The accelerator_view that the array is to be created on.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| D3d_buffer | <p>AN AMP supported D3D device buffer pointer to be used to create the array. The parameter must meet all of the following conditions for successful creation of a accelerator_view:</p> <ol style="list-style-type: none"> <li>1) Must be a supported D3D buffer interface. For this release, only ID3D11Buffer interface is supported.</li> <li>2) The D3D device on which the buffer was created must be the same as that underlying the accelerator_view parameter <i>av</i>.</li> <li>3) The D3D buffer must additionally satisfy the following conditions: <ul style="list-style-type: none"> <li>a. The buffer size in bytes must be greater than or equal to the size in bytes of the field to be created (<i>g.get_size() * sizeof(elem_type)</i>).</li> <li>b. Must not have been created with D3D11_USAGE_STAGING.</li> <li>c. SHADER_RESOURCE and/or UNORDERED_ACCESS bindings should be allowed for the buffer.</li> <li>d. Raw views must be allowed for the buffer (e.g. D3D11_RESOURCE_MISC_BUFFER_ALLOW_RAW_VIEWS).</li> </ul> </li> </ol> |

**Return Value:**

The newly created array object.

**Exceptions:**

|                                                        |                                |
|--------------------------------------------------------|--------------------------------|
| Invalid extents argument                               | concurrency::runtime_exception |
| NULL D3D buffer pointer                                | concurrency::runtime_exception |
| Invalid D3D buffer argument.                           | concurrency::runtime_exception |
| Cannot create D3D buffer on a non-D3D accelerator_view | concurrency::runtime_exception |

5096  
5097

```
template <typename T, int N> IUnknown * get_buffer(const array<T, N>& f);
```

Returns a D3D buffer interface pointer underlying the passed array. Fails with a “*runtime\_exception*” exception if the passed array is not on a D3D accelerator view. On success, it increments the reference count of the D3D buffer interface by calling “*AddRef*” on the interface. Users must call “*Release*” on the returned interface after they are finished using it, for proper reclamation of the resources associated with the object.

|                                                                                                                                                                                                                           |                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| <b>Parameters:</b>                                                                                                                                                                                                        |                                                                    |
| <i>f</i>                                                                                                                                                                                                                  | The array for which the underlying D3D buffer interface is needed. |
| <b>Return Value:</b>                                                                                                                                                                                                      |                                                                    |
| A IUnknown interface pointer corresponding to the D3D buffer underlying the passed array. Users must use the QueryInterface member function on the returned interface to obtain the correct D3D buffer interface pointer. |                                                                    |
| <b>Exceptions:</b>                                                                                                                                                                                                        |                                                                    |
| Cannot get D3D buffer from a non-D3D array                                                                                                                                                                                | concurrency::runtime_exception                                     |

5098

**void d3d\_access\_lock(accelerator\_view& av)**

Acquires a non-recursive lock on the internal mutex used by the *accelerator\_view* to synchronize operations. No operations on the provided *accelerator\_view* or on data structures associated with it will proceed while this lock is held, allowing other threads to make use of Direct3D resources shared between the application and C++ AMP without race conditions. It is undefined behavior to lock an *accelerator\_view* from a thread that already holds the lock, or to perform any operations on the *accelerator\_view* or data structures associated with the *accelerator\_view* from the thread that holds the lock.

This function will block until the lock is acquired.

**Parameters:**

|           |                                      |
|-----------|--------------------------------------|
| <i>av</i> | The <i>accelerator_view</i> to lock. |
|-----------|--------------------------------------|

5099

**bool d3d\_access\_try\_lock(accelerator\_view& av)**

Attempt to lock the provided *accelerator\_view* without blocking.

**Parameters:**

|           |                                      |
|-----------|--------------------------------------|
| <i>av</i> | The <i>accelerator_view</i> to lock. |
|-----------|--------------------------------------|

**Return Value:**

true if the lock was acquired, or false if the lock was held by another thread at the time of the call.

5100

**void d3d\_access\_unlock(accelerator\_view& av)**

Releases the lock on the provided *accelerator\_view*. It is undefined behavior to call *d3d\_access\_lock* from a thread that does not hold the lock.

**Parameters:**

|           |                                        |
|-----------|----------------------------------------|
| <i>av</i> | The <i>accelerator_view</i> to unlock. |
|-----------|----------------------------------------|

5101

**11.1 scoped\_d3d\_access\_lock**

The *scoped\_d3d\_access\_lock* class provides an RAII-style wrapper around the *d3d\_access\_lock* and *d3d\_access\_unlock* functions. Scoped locks cannot be copied but can be moved. Struct *adopt\_d3d\_access\_lock\_t* is a tag type to indicate the D3D access lock should be adopted rather than acquired.

**5106 11.1.1 Synopsis**

5107  
5108 **struct adopt\_d3d\_access\_lock\_t {};**  
5109

```

5110 class scoped_d3d_access_lock
5111 {
5112     public:
5113     explicit scoped_d3d_access_lock(accelerator_view& av);
5114     explicit scoped_d3d_access_lock(accelerator_view& av, adopt_d3d_access_lock_t t);
5115     scoped_d3d_access_lock(scoped_d3d_access_lock&& other);
5116     scoped_d3d_access_lock& operator=(scoped_d3d_access_lock&& other);
5117     ~scoped_d3d_access_lock();
5118 };

```

## 5119 11.1.2 Constructors

5120

|                                                                |  |
|----------------------------------------------------------------|--|
| <code>scoped_d3d_access_lock(accelerator_view&amp; av);</code> |  |
|----------------------------------------------------------------|--|

Constructs a `scoped_d3d_access_lock`, acquiring the lock on the provided `accelerator_view`.

**Parameters**

|                 |                                            |
|-----------------|--------------------------------------------|
| <code>av</code> | The <code>accelerator_view</code> to lock. |
|-----------------|--------------------------------------------|

5121

|                                                                                           |  |
|-------------------------------------------------------------------------------------------|--|
| <code>scoped_d3d_access_lock(accelerator_view&amp; av, adopt_d3d_access_lock_t t);</code> |  |
|-------------------------------------------------------------------------------------------|--|

Constructs a `scoped_d3d_access_lock` object without acquiring the lock on `av`. It is assumed that the lock is already held by the calling thread (e.g. acquired through `d3d_access_try_lock`).

**Parameters**

|                 |                                                                             |
|-----------------|-----------------------------------------------------------------------------|
| <code>av</code> | The <code>accelerator_view</code> to unlock when this object is destructed. |
|-----------------|-----------------------------------------------------------------------------|

|                |                                                                        |
|----------------|------------------------------------------------------------------------|
| <code>t</code> | An object of the tag class type <code>adopt_d3d_access_lock_t</code> . |
|----------------|------------------------------------------------------------------------|

5122

## 5123 11.1.3 Move constructors and assignment operators

5124

|                                                                                             |  |
|---------------------------------------------------------------------------------------------|--|
| <code>scoped_d3d_access_lock(scoped_d3d_access_lock&amp;&amp; other);</code>                |  |
| <code>scoped_d3d_access_lock&amp; operator=(scoped_d3d_access_lock&amp;&amp; other);</code> |  |

`scoped_d3d_access_lock` objects cannot be copied but can be moved. Lock ownership is transferred from `other` to the object being constructed or overwritten. Any lock previously held by the object being assigned into is released.

**Parameters**

|                    |                                                          |
|--------------------|----------------------------------------------------------|
| <code>other</code> | Source <code>scoped_d3d_access_lock</code> to move from. |
|--------------------|----------------------------------------------------------|

5125

## 5126 11.1.4 Destructor

5127

|                                         |  |
|-----------------------------------------|--|
| <code>~scoped_d3d_access_lock();</code> |  |
|-----------------------------------------|--|

Releases the lock held by the `scoped_d3d_access_lock`, if any.

# 5128 12 Error Handling

5129

## 5130 12.1 static\_assert

5131

The C++ intrinsic `static_assert` is often used to handle error states that are detectable at compile time. In this way `static_assert` is a technique for conveying static semantic errors and as such they will be categorized similar to exception types.

5135

5136 **12.2 Runtime errors**

5137

5138 On encountering an irrecoverable error, C++ AMP runtime throws a C++ exception to communicate/propagate the error to  
 5139 client code. (Note: exceptions are not thrown from `restrict(amp)` code.) The actual exceptions thrown by each API are  
 5140 listed in the API descriptions. Following are the exception types thrown by C++ AMP runtime:

5141

5142 **12.2.1 runtime\_exception**

5143

5144 The exception type that all AMP runtime exceptions derive from. A `runtime_exception` instance comprises a textual  
 5145 description of the error and a `HRESULT` error code to indicate the cause of the error.

5146

5147 **12.2.1.1 Synopsis**

```
5148 class runtime_exception : public std::exception
5149 {
5150 public:
5151     runtime_exception(const char * message, HRESULT hresult) throw();
5152     explicit runtime_exception(HRESULT hresult) throw();
5153     runtime_exception(const runtime_exception& other) throw();
5154     runtime_exception& operator=(const runtime_exception& other) throw();
5155     virtual ~runtime_exception() throw();
5156     HRESULT get_error_code() const throw();
5157 };
5158
```

5159 **12.2.1.2 Constructors**

5160

|                                                                               |
|-------------------------------------------------------------------------------|
| <code>runtime_exception(const char* message, HRESULT hresult) throw();</code> |
|-------------------------------------------------------------------------------|

Construct a `runtime_exception` exception with the specified message and `HRESULT` error code.

**Parameters:**

|                      |                              |
|----------------------|------------------------------|
| <code>message</code> | Descriptive message of error |
|----------------------|------------------------------|

|                      |                                                            |
|----------------------|------------------------------------------------------------|
| <code>hresult</code> | <code>HRESULT</code> error code that caused this exception |
|----------------------|------------------------------------------------------------|

5161

5162

|                                                           |
|-----------------------------------------------------------|
| <code>runtime_exception (HRESULT hresult) throw();</code> |
|-----------------------------------------------------------|

Construct a `runtime_exception` exception with the specified `HRESULT` error code.

**Parameters:**

|                      |                                                            |
|----------------------|------------------------------------------------------------|
| <code>hresult</code> | <code>HRESULT</code> error code that caused this exception |
|----------------------|------------------------------------------------------------|

5163

|                                                                             |
|-----------------------------------------------------------------------------|
| <code>runtime_exception(const runtime_exception&amp; other) throw();</code> |
|-----------------------------------------------------------------------------|

|                                                                                            |
|--------------------------------------------------------------------------------------------|
| <code>runtime_exception &amp;operator=(const runtime_exception&amp; other) throw();</code> |
|--------------------------------------------------------------------------------------------|

Copy constructor and assignment operator

**Parameters:**

|                      |                                                            |
|----------------------|------------------------------------------------------------|
| <code>hresult</code> | <code>HRESULT</code> error code that caused this exception |
|----------------------|------------------------------------------------------------|

5164

5165 12.2.1.3 Members

5166

|                                                     |
|-----------------------------------------------------|
| <code>HRESULT get_error_code() const throw()</code> |
|-----------------------------------------------------|

Returns the error code that caused **this** exception.

**Return Value:**

Returns the HRESULT error code that caused **this** exception.

5167

|                                                   |
|---------------------------------------------------|
| <code>virtual ~runtime_exception() throw()</code> |
|---------------------------------------------------|

Destruct a runtime\_exception exception object instance.

5168

5169 12.2.1.4 Specific Runtime Exceptions

| Exception String                    | Source                                     | Explanation                                                                         |
|-------------------------------------|--------------------------------------------|-------------------------------------------------------------------------------------|
| No supported accelerator available. | Accelerator constructor, array constructor | No device available at runtime supports C++ AMP.                                    |
| Failed to create buffer             | Array constructor                          | Couldn't create buffer on accelerator, likely due to lack of resource availability. |
|                                     |                                            |                                                                                     |
|                                     |                                            |                                                                                     |
|                                     |                                            |                                                                                     |
|                                     |                                            |                                                                                     |

5170

5171 12.2.2 `out_of_memory`

5172

5173 An instance of this exception type is thrown when an underlying OS API call fails due to failure to allocate system or device  
 5174 memory (*E\_OUTOFMEMORY* *HRESULT* error code). Note that if the runtime fails to allocate memory from the heap using  
 5175 the C++ *new* operator, a *std::bad\_alloc* exception is thrown and not the C++ AMP *out\_of\_memory* exception.  
 5176

5177 12.2.2.1 Synopsis

```
5178 class out_of_memory : public runtime_exception
5179 {
520 public:
521     explicit out_of_memory(const char * message) throw();
522     out_of_memory () throw();
523 };
524
```

5185 12.2.2.2 Constructor

5186

|                                                                   |
|-------------------------------------------------------------------|
| <code>explicit out_of_memory(const char * message) throw()</code> |
|-------------------------------------------------------------------|

Construct a `out_of_memory` exception with the specified message.

**Parameters:**

|                      |                              |
|----------------------|------------------------------|
| <code>message</code> | Descriptive message of error |
|----------------------|------------------------------|

5187

5188

|                                      |
|--------------------------------------|
| <code>out_of_memory() throw()</code> |
|--------------------------------------|

Construct a `out_of_memory` exception.

**Parameters:**

|       |
|-------|
| None. |
|-------|

5189 **12.2.3 invalid\_compute\_domain**

5190

5191 An instance of this exception type is thrown when the runtime fails to devise a dispatch for the compute domain specified  
 5192 at a *parallel\_for\_each* call site.  
 5193

5194 **12.2.3.1 Synopsis**

```
5195 class invalid_compute_domain : public runtime_exception
5196 {
5197     public:
5198         explicit invalid_compute_domain (const char * message) throw();
5199         invalid_compute_domain () throw();
5200     };
5201
```

5202 **12.2.3.2 Constructor**

5203

|                                                                      |
|----------------------------------------------------------------------|
| <b>explicit invalid_compute_domain(const char * message) throw()</b> |
|----------------------------------------------------------------------|

Construct an invalid\_compute\_domain exception with the specified message.

**Parameters:**

|         |                              |
|---------|------------------------------|
| message | Descriptive message of error |
|---------|------------------------------|

5204

5205

|                                         |
|-----------------------------------------|
| <b>invalid_compute_domain() throw()</b> |
|-----------------------------------------|

Construct an invalid\_compute\_domain exception.

**Parameters:**

|       |
|-------|
| None. |
|-------|

5206

5207 **12.2.4 unsupported\_feature**

5208

5209 An instance of this exception type is thrown on executing a *restrict(amp)* function on the host which uses an intrinsic  
 5210 unsupported on the host (such as *tiled\_index<>::barrier.wait()*) or when invoking a *parallel\_for\_each* or allocating an object  
 5211 on an accelerator which doesn't support certain features which are required for the execution to proceed, such as, but not  
 5212 limited to:  
 5213

- 5214 1. The accelerator is not capable of executing code, but serves as a memory allocation arena only
- 5215 2. The accelerator doesn't support the allocation of textures
- 5216 3. A texture object is created with an invalid combination of bits\_per\_scalar\_element and short-vector type
- 5217 4. Read and write operations are both requested on a texture object with bits\_per\_scalar != 32

5218 **12.2.4.1 Synopsis**

```
5219 class unsupported_feature : public runtime_exception
5220 {
5221     public:
5222         explicit unsupported_feature (const char * message) throw();
5223         unsupported_feature () throw();
5224     };
```

5225

## 5226 12.2.4.2 Constructor

5227

|                                                                   |
|-------------------------------------------------------------------|
| <code>class unsupported_feature : public runtime_exception</code> |
|-------------------------------------------------------------------|

|                                                       |
|-------------------------------------------------------|
| Exception thrown when an unsupported feature is used. |
|-------------------------------------------------------|

5228

|                                                                         |
|-------------------------------------------------------------------------|
| <code>explicit unsupported_feature (const char* message) throw()</code> |
|-------------------------------------------------------------------------|

|                                                                        |
|------------------------------------------------------------------------|
| Construct an unsupported_feature exception with the specified message. |
|------------------------------------------------------------------------|

|                    |
|--------------------|
| <b>Parameters:</b> |
|--------------------|

|                      |                              |
|----------------------|------------------------------|
| <code>message</code> | Descriptive message of error |
|----------------------|------------------------------|

5229

5230

|                                            |
|--------------------------------------------|
| <code>unsupported_feature() throw()</code> |
|--------------------------------------------|

|                                             |
|---------------------------------------------|
| Construct an unsupported_feature exception. |
|---------------------------------------------|

|                    |
|--------------------|
| <b>Parameters:</b> |
|--------------------|

|       |
|-------|
| None. |
|-------|

5231

## 5232 12.2.5 accelerator\_view\_removed

5233

An instance of this exception type is thrown when the C++ AMP runtime detects that a connection with a particular accelerator, represented by an instance of class `accelerator_view`, has been lost. When such an incident happens, all data allocated through the accelerator view and all in-progress computations on the accelerator view may be lost. This exception may be thrown by `parallel_for_each`, as well as any other copying and/or synchronization method.

5234

## 5235 12.2.5.1 Synopsis

```
5240 class accelerator_view_removed : public runtime_exception
5241 {
5242     public:
5243         explicit accelerator_view_removed(const char* message, HRESULT view_removed_reason)
5244             throw();
5245         explicit accelerator_view_removed(HRESULT view_removed_reason) throw();
5246         HRESULT get_view_removed_reason() const throw();
5247     };
5248 }
```

## 5249 12.2.5.2 Constructor

|                                                                                                           |
|-----------------------------------------------------------------------------------------------------------|
| <code>explicit accelerator_view_removed(const char* message, HRESULT view_removed_reason) throw();</code> |
|-----------------------------------------------------------------------------------------------------------|

|                                                                                      |
|--------------------------------------------------------------------------------------|
| <code>explicit accelerator_view_removed(HRESULT view_removed_reason) throw();</code> |
|--------------------------------------------------------------------------------------|

|                                                                                        |
|----------------------------------------------------------------------------------------|
| Construct an accelerator_view_removed exception with the specified message and HRESULT |
|----------------------------------------------------------------------------------------|

|                    |
|--------------------|
| <b>Parameters:</b> |
|--------------------|

|                      |                              |
|----------------------|------------------------------|
| <code>message</code> | Descriptive message of error |
|----------------------|------------------------------|

|                                  |                                                                                         |
|----------------------------------|-----------------------------------------------------------------------------------------|
| <code>view_removed_reason</code> | HRESULT error code indicating the cause of removal of the <code>accelerator_view</code> |
|----------------------------------|-----------------------------------------------------------------------------------------|

5250

## 5251 12.2.5.3 Members

5252

`HRESULT get_view_removed_reason() const throw();`

Provides the HRESULT error code indicating the cause of removal of the accelerator\_view

**Return Value:**

The HRESULT error code indicating the cause of removal of the accelerator\_view

5253

## 12.3 Error handling in device code (amp-restricted functions) (Optional)

5255

5256 The use of the `throw` C++ keyword is disallowed in C++ AMP vector functions (*amp* restricted) and will result in a  
5257 compilation error. C++ AMP offers the following intrinsics in vector code for error handling.

5258

5259 **Microsoft-specific:** the Microsoft implementation of C++ AMP provides the methods specified in this section, provided all of  
5260 the following conditions are met.

- 5261 1. The debug version of the runtime is being used (i.e. the code is compiled with the `_DEBUG` preprocessor definition).
- 5262 2. The debug layer is available on the system. This, in turn requires DirectX SDK to be installed on the system on  
5263 Windows 7. On Windows 8 no SDK installation is necessary..
- 5264 3. The accelerator\_view on which the kernel is invoked must be on a device which supports the `printf` and `abort`  
5265 intrinsics. As of the date of writing this document, only the REF device supports these intrinsics.

5266

5267 When the debug version of the runtime is not used or the debug layer is unavailable, executing a kernel that using these  
5268 intrinsics through a `parallel_for_each` call will result in a runtime exception. On devices that do not support these intrinsics,  
5269 these intrinsics will behave as no-ops.

5270

`void direct3d_printf(const char* format_string, ...) restrict(amp)`

Prints formatted output from a kernel to the debug output. The formatting semantics are same as the C Library `printf` function. Also, this function is executed as any other device-side function: per-thread, and in the context of the calling thread. Due to the asynchronous nature of kernel execution, the output from this call may appear anytime between the launch of the kernel containing the `printf` call and completion of the kernel's execution.

**Parameters:**

|                            |                                                   |
|----------------------------|---------------------------------------------------|
| <code>format_string</code> | The format string.                                |
| ...                        | An optional list of parameters of variable count. |

**Return Value:**

None.

5271

`void direct3d_errorf(const char* format_string, ...) restrict(amp)`

This intrinsic prints formatted error messages from a kernel to the debug output. This function is executed as any other device-side function: per-thread, and in the context of the calling thread. Note that due to the asynchronous nature of kernel execution, the actual error messages may appear in the debug output asynchronously, any time between the dispatch of the kernel and the completion of the kernel's execution. When these error messages are detected by the runtime, it raises a "runtime\_exception" exception on the host with the formatted error message output as the exception message.

**Parameters:**

|                            |                                                   |
|----------------------------|---------------------------------------------------|
| <code>format_string</code> | The format string.                                |
| ...                        | An optional list of parameters of variable count. |

5272

`void direct3d_abort() restrict(amp)`

This intrinsic aborts the execution of threads in the compute domain of a kernel invocation, that execute this instruction. This function is executed as any other device-side function: per-thread, and in the context of the calling thread. Also the thread is terminated without executing any destructors for local variables. When the abort is detected by the runtime, it

raises a “`runtime_exception`” exception on the host with the abort output as the exception message. Note that due to the asynchronous nature of kernel execution, the actual abort may be detected any time between the dispatch of the kernel and the completion of the kernel’s execution.

5273  
 5274  
 5275 Due to the asynchronous nature of kernel execution, the `direct3d_printf`, `direct3d_errorf` and `direct3d_abort` messages from  
 5276 kernels executing on a device appear asynchronously during the execution of the shader or after its completion and not  
 5277 immediately after the async launch of the kernel. Thus these messages from a kernel may be interleaved with messages  
 5278 from other kernels executing concurrently or error messages from other runtime calls in the debug output. It is the  
 5279 programmer’s responsibility to include appropriate information in the messages originating from kernels to indicate the  
 5280 origin of the messages.

## 5281 13 Appendix: C++ AMP Future Directions (Informative)

5282  
 5283 It is likely that C++ AMP will evolve over time. The set of features allowed inside `amp`-restricted functions will grow.  
 5284 However, compilers will have to continue to support older hardware targets which only support the previous, smaller  
 5285 feature set. This section outlines possible such evolution of the language syntax and associated feature set.  
 5286

### 5287 13.1 Versioning Restrictions

5288 This section contains an informative description of additional language syntax and rules to allow the versioning of C++ AMP  
 5289 code. If an implementation desires to extend C++ AMP in a manner not covered by this version of the specification, it is  
 5290 recommended that it follows the syntax and rules specified here.

#### 5291 13.1.1 `auto` restriction

5292 The `restriction` production (section 2.1) of the C++ grammar is amended to allow the contextual keyword `auto`.  
 5293

5294     `restriction:`  
 5295         `amp-restriction`  
 5296         `cpu`  
 5297         `auto`

5298 A function or lambda which is annotated with `restrict(auto)` directs the compiler to check all known restrictions and  
 5299 automatically deduce the set of restrictions that a function complies with. `restrict(auto)` is only allowed for functions where  
 5300 the function declaration is also a function definition, and no other declaration of the same function occurs.  
 5301

5302 A function may be simultaneously explicitly and `auto` restricted, e.g., `restrict(cpu,auto)`. In such case, it will be explicitly  
 5303 checked for compulsory conformance with the set of explicitly specified (non-auto) restrictions, and implicitly checked for  
 5304 possible conformance with all other restrictions that the compiler supports.  
 5305

5306 Consider the following example:  
 5307

```
5308     int f1() restrict(amp);  

  5309  

  5310     int f2() restrict(cpu,auto)  

  5311     {  

  5312         f1();  

  5313     }  

  5314
```

5315 In this example, `f2` is verified for compulsory adherence to the `restrict(cpu)` restriction. This results in an error, since `f2` calls  
 5316 `f1`, which is not `cpu`-restricted. Had we changed `f1`’s restriction to `restrict(cpu)`, then `f2` will pass the adherence test to the  
 5317 explicitly specified `restrict(cpu)`. Now with respect to the `auto` restriction, the compiler has to check whether `f2` conforms to  
 5318 `restrict(amp)`, which is the only other restriction not explicitly specified. In the context of verifying the plausibility of  
 5319

5320 inferring an *amp*-restriction for *f2*, the compiler notices that *f2* calls *f1*, which is, in our modified example, not *amp*-  
 5321 restricted, and therefore *f2* is also inferred to be not *amp*-restricted. Thus the total inferred restriction for *f2* is *restrict(cpu)*.  
 5322 If we now change the restriction for *f1* into *restrict(cpu,amp)*, then the inference for *f2* would reach the conclusion that *f2*  
 5323 is *restrict(cpu,amp)* too.

5324  
 5325 When two overloads are available to call from a given restriction context, and they differ only by the fact that one is  
 5326 explicitly restricted while the other is implicitly inferred to be restricted, the explicitly restricted overload shall be chosen.

### 5327 13.1.2 Automatic restriction deduction

5328 Implementations are encouraged to support a mode in which functions that have their definitions accompany their  
 5329 declarations, and where no other declarations occur for such functions, have their restriction set automatically deduced.

5330  
 5331 In such a mode, when the compiler encounters a function declaration which is also a definition, and a previous declaration  
 5332 for the function hasn't been encountered before, then the compiler analyses the function as if it was restricted with  
 5333 *restrict(cpu,auto)*. This allows easy reuse of existing code in *amp*-restricted code, at the cost of prolonged compilation times.

### 5334 13.1.3 amp Version

5335 The *amp*-restriction production of the C++ grammar is amended thus:

5336  
 5337     *amp-restriction*:  
 5338         **amp** *amp-version*<sub>opt</sub>  
 5339  
 5340     *amp-version*:  
 5341         : *integer-constant*  
 5342         : *integer-constant* . *integer-constant*

5343 An *amp* version specifies the lowest version of amp that this function supports. In other words, if a function is decorated  
 5344 with *restrict(amp:1)*, then that function also supports any version greater or equal to 1. When the *amp* version is elided,  
 5345 the implied version is implementation-defined. Implementations are encouraged to support a compiler flag controlling the  
 5346 default version assumed. When versioning is used in conjunction with *restrict(auto)* and/or automatic restriction deduction,  
 5347 the compiler shall infer the maximal version of the *amp* restriction that the function adheres to.

5348  
 5349 Section 2.3.2 specifies that restriction specifiers of a function shall not overlap with any restriction specifiers in another  
 5350 function within the same overload set.

```
5351     int func(int x) restrict(cpu,amp);
  5352     int func(int x) restrict(cpu); // error, overlaps with previous declaration
```

5353 This rule is relaxed in the case of versioning: functions overloaded with *amp* versions are not considered to overlap:

```
5354     int func(int x) restrict(cpu);
  5355     int func(int x) restrict(amp:1);
  5356     int func(int x) restrict(amp:2);
```

5357 When an overload set contains multiple versions of the amp specifier, the function with the highest version number that is  
 5358 not higher than the callee is chosen:

```
5359     void glorp() restrict(amp:1) { }
  5360     void glorp() restrict(amp:2) { }

  5361     void glorp_caller() restrict(amp:2) {
  5362         glop(); // okay; resolves to call "glop() restrict(amp:2)"
  5363     }
```

## 5371 13.2 Projected Evolution of *amp*-Restricted Code

5372 Based on the nascent availability of features in advanced GPUs and corresponding hardware-vendor-specific programming  
 5373 models, it is apparent that the limitations associated with *restrict(amp)* will be gradually lifted. The table below captures  
 5374 one possible path for future *amp* versions to follow. If implementers need to (non-normatively) extend the *amp*-restricted  
 5375 language subset, it is recommended that they consult the table below and try to conform to its style.

5376 Implementations may not define an amp version greater or equal to 2.0. All non-normative extensions shall be restricted to  
 5377 the patterns 1.x (where x > 0). Version number 1.0 is reserved to implementations strictly adhering to this version of the  
 5378 specification, while version number 2.0 is reserved for the next major version of this specification.

| Area                        | Feature                          | amp:1             | amp:1.1 | amp:1.2 | amp:2 | cpu |
|-----------------------------|----------------------------------|-------------------|---------|---------|-------|-----|
| Local/Param/Function Return | char (8 - signed/unsigned/plain) | No                | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | short (16 - signed/unsigned)     | No                | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | int (32 - signed/unsigned)       | Yes               | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | long (32 - signed/unsigned)      | Yes               | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | long long (64 - signed/unsigned) | No                | No      | Yes     | Yes   | Yes |
| Local/Param/Function Return | half-precision float (16)        | No                | No      | No      | No    | No  |
| Local/Param/Function Return | float (32)                       | Yes               | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | double (64)                      | Yes <sup>10</sup> | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | long double (?)                  | No                | No      | No      | No    | Yes |
| Local/Param/Function Return | bool (8)                         | Yes               | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | wchar_t (16)                     | No                | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | Pointer (single-indirection)     | Yes               | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | Pointer (multiple-indirection)   | No                | No      | Yes     | Yes   | Yes |
| Local/Param/Function Return | Reference                        | Yes               | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | Reference to pointer             | Yes               | Yes     | Yes     | Yes   | Yes |
| Local/Param/Function Return | Reference/pointer to function    | No                | No      | Yes     | Yes   | Yes |
| Local/Param/Function Return | static local                     | No                | No      | Yes     | Yes   | Yes |
| Struct/class/union members  | char (8 - signed/unsigned/plain) | No                | Yes     | Yes     | Yes   | Yes |
| Struct/class/union members  | short (16 - signed/unsigned)     | No                | Yes     | Yes     | Yes   | Yes |
| Struct/class/union members  | int (32 - signed/unsigned)       | Yes               | Yes     | Yes     | Yes   | Yes |
| Struct/class/union members  | long (32 - signed/unsigned)      | Yes               | Yes     | Yes     | Yes   | Yes |
| Struct/class/union members  | long long (64 - signed/unsigned) | No                | No      | Yes     | Yes   | Yes |
| Struct/class/union members  | half-precision float (16)        | No                | No      | No      | No    | No  |
| Struct/class/union members  | float (32)                       | Yes               | Yes     | Yes     | Yes   | Yes |
| Struct/class/union members  | double (64)                      | Yes               | Yes     | Yes     | Yes   | Yes |
| Struct/class/union members  | long double (?)                  | No                | No      | No      | No    | Yes |
| Struct/class/union members  | bool (8)                         | No                | Yes     | Yes     | Yes   | Yes |
| Struct/class/union members  | wchar_t (16)                     | No                | Yes     | Yes     | Yes   | Yes |
| Struct/class/union members  | Pointer                          | No                | No      | Yes     | Yes   | Yes |
| Struct/class/union members  | Reference                        | No                | No      | Yes     | Yes   | Yes |

<sup>10</sup> Double precision support is an optional feature on some amp:1-compliant hardware.

|                            |                                         |     |     |     |     |     |
|----------------------------|-----------------------------------------|-----|-----|-----|-----|-----|
| Struct/class/union members | Reference/pointer to function           | No  | No  | No  | Yes | Yes |
| Struct/class/union members | bitfields                               | No  | No  | No  | Yes | Yes |
| Struct/class/union members | unaligned members                       | No  | No  | No  | No  | Yes |
| Struct/class/union members | pointer-to-member (data)                | No  | No  | Yes | Yes | Yes |
| Struct/class/union members | pointer-to-member (function)            | No  | No  | Yes | Yes | Yes |
| Struct/class/union members | static data members                     | No  | No  | No  | Yes | Yes |
| Struct/class/union members | static member functions                 | Yes | Yes | Yes | Yes | Yes |
| Struct/class/union members | non-static member functions             | Yes | Yes | Yes | Yes | Yes |
| Struct/class/union members | Virtual member functions                | No  | No  | Yes | Yes | Yes |
| Struct/class/union members | Constructors                            | Yes | Yes | Yes | Yes | Yes |
| Struct/class/union members | Destructors                             | Yes | Yes | Yes | Yes | Yes |
| Enums                      | char (8 - signed/unsigned/plain)        | No  | Yes | Yes | Yes | Yes |
| Enums                      | short (16 - signed/unsigned)            | No  | Yes | Yes | Yes | Yes |
| Enums                      | int (32 - signed/unsigned)              | Yes | Yes | Yes | Yes | Yes |
| Enums                      | long (32 - signed/unsigned)             | Yes | Yes | Yes | Yes | Yes |
| Enums                      | long long (64 - signed/unsigned)        | No  | No  | No  | No  | Yes |
| Structs/Classes            | Non-virtual base classes                | Yes | Yes | Yes | Yes | Yes |
| Structs/Classes            | Virtual base classes                    | No  | Yes | Yes | Yes | Yes |
| Arrays                     | of pointers                             | No  | No  | Yes | Yes | Yes |
| Arrays                     | of arrays                               | Yes | Yes | Yes | Yes | Yes |
| Declarations               | tile_static                             | Yes | Yes | Yes | Yes | No  |
| Function Declarators       | Varargs (...)                           | No  | No  | No  | No  | Yes |
| Function Declarators       | throw() specification                   | No  | No  | No  | No  | Yes |
| Statements                 | global variables                        | No  | No  | No  | Yes | Yes |
| Statements                 | static class members                    | No  | No  | No  | Yes | Yes |
| Statements                 | Lambda capture-by-reference (on gpu)    | No  | No  | Yes | Yes | Yes |
| Statements                 | Lambda capture-by-reference (in p_f_e)  | No  | No  | No  | Yes | Yes |
| Statements                 | Recursive function call                 | No  | No  | Yes | Yes | Yes |
| Statements                 | conversion between pointer and integral | No  | Yes | Yes | Yes | Yes |
| Statements                 | new                                     | No  | No  | Yes | Yes | Yes |
| Statements                 | delete                                  | No  | No  | Yes | Yes | Yes |
| Statements                 | dynamic_cast                            | No  | No  | No  | No  | Yes |
| Statements                 | typeid                                  | No  | No  | No  | No  | Yes |
| Statements                 | goto                                    | No  | No  | No  | No  | Yes |
| Statements                 | labels                                  | No  | No  | No  | No  | Yes |
| Statements                 | asm                                     | No  | No  | No  | No  | Yes |
| Statements                 | throw                                   | No  | No  | No  | No  | Yes |
| Statements                 | try/catch                               | No  | No  | No  | No  | Yes |
| Statements                 | __try/__except                          | No  | No  | No  | No  | Yes |
| Statements                 | __leave                                 | No  | No  | No  | No  | Yes |

5381

5382