OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems, and handheld devices.

Specification documents and online reference are available at www.khronos.org/opencl.

OpenCL API Reference

Section and table references are to the OpenCL API 2.1 specification.

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices. Items in blue apply when the appropriate extension is supported.

Querying Platform Info & Devices [4.1-4.2] [9.16.9]

- `clGetPlatformIDs` (cl_uint num_entries, cl_platform_id *platforms, cl_uint *num_platforms)
- `clGetPlatformInfo` (cl_platform_id platform, param_name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)
- `clGetDeviceIDs` (cl_platform_id platform, device_type, cl_uint num_entries, cl_device_id *devices, cl_uint *num_devices)
- `clGetDeviceInfo` (cl_device_id device, device_info_info param_name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)
- `clGetDeviceAndHostTimer` (cl_device_id_device, cl_ulong *device_timestamp, cl_ulong *host_timestamp)

Partitioning a Device [4.3]

- `clCreateSubDevices` (cl_device_id_device, cl_device_id*, cl_device_partition_property* properties, cl_uint num_devices, cl_device_id* out_devices, cl_uint* num_devices_ret)

The OpenCL Runtime

API calls that manage OpenCL objects such as command-queues, memory objects, program objects, kernel objects for __kernel functions in a program and calls that allow you to enqueue commands to a command-queue such as executing a kernel, reading, or writing a memory object.

Command Queues [5.1]

- `clCreateCommandQueueWithProperties` (cl_context context, cl_device_id_device, const cl_queue_properties* properties, cl_int *errcode_ret)
- `clGetCommandQueue` (cl_command_queue queue, cl_context* ctx, cl_device_id* device)

Contexts [4.4]

- `clCreateContext` (const cl_context_properties* properties, cl_uint num_devices, const cl_device_id_device*, void* cbfn_notify)
- `clReleaseContext` (cl_context context)

Get CL Extension Function Pointers [9.2]

- `void* clGetExtensionFunctionAddressForPlatform` (cl_platform_id_platform, const char* funcname)
OpenCL Class Diagram

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language\(^1\) (UML) notation. The diagram shows both nodes and edges which are classes and their relationships. As a simplification it shows only classes, and no attributes or operations.

### Annotations

- **Cardinality:**
  - Many
  - One and only one
  - Optionally one
  - One or more

### Interactions

- **Pipe Object Queries**
  - Create Pipe Objects
  - Get Pipe Info
  - Get Memory Object Info
  - Get Feature Info
  - Get Command Queue Info

### Pipes

A pipe is a memory object that stores data organized as a FIFO. Pipe objects can only be accessed using built-in functions that read from and write to a pipe. Pipe objects are not accessible from the host.

### Memory Objects

- **Memory Object**
  - A memory object is a handle to a reference counted region of global memory.

### Create Buffer Objects

- **clCreateBuffer**
  - **flags:**
    - CL_MEM_READ_WRITE
    - CL_MEM_{READ, WRITE}
    - CL_MEM_{READ, WRITE}_ONLY

### Read, Write, Copy, Fill Buffer Objects

- **clEnqueueReadBuffer**
- **clEnqueueWriteBuffer**
- **clEnqueueFillBuffer**
- **clEnqueueCopyBuffer**
- **clEnqueueCopyBufferRect**

### Conversions and Type Casting Examples

\[ T \Rightarrow T; \Rightarrow Scalar to scalar; \Rightarrow \text{or scalar to vector} \]

- **T \Rightarrow T**
- **T \Rightarrow T**
- **T \Rightarrow T**
- **T \Rightarrow T**
- **T \Rightarrow T**

### Map Buffer Objects

- **clEnqueueMapBuffer**

### Memory Objects

- **Memory Object**
  - A memory object is a handle to a reference counted region of global memory.

### Buffer Objects

- **Elements of buffer objects are stored sequentially and accessed using a pointer by a kernel executing on a device.**

### Create Buffer Objects

- **clCreateBuffer**
  - **flags:**
    - CL_MEM_READ_WRITE
    - CL_MEM_{READ, WRITE}
    - CL_MEM_{READ, WRITE}_ONLY

### OpenCL Device Architecture Diagram

The table below shows memory regions with allocation and memory access capabilities.

<table>
<thead>
<tr>
<th>Host</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>No allocation</td>
<td>R/W access</td>
</tr>
<tr>
<td>R/W access</td>
<td>Static allocation</td>
</tr>
<tr>
<td>Static allocation</td>
<td>R/W access</td>
</tr>
<tr>
<td>R/W access</td>
<td>No allocation</td>
</tr>
</tbody>
</table>

### Pipelines

A pipe is a memory object that stores data organized as a FIFO. Pipe objects can only be accessed using built-in functions that read from and write to a pipe. Pipe objects are not accessible from the host.

### Create Pipe Objects

- **Create Pipe Objects**
  - **flags:**
    - CL_PIPE_MAX_PACKETS

### Memory Objects

- **Memory Object**
  - A memory object is a handle to a reference counted region of global memory.

### Query Memory Object

- **Query Memory Object**
  - **flags:**
    - CL_MIGRATE_MEM_OBJECT_HOST
    - CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

---

\(^1\) Unified Modeling Language (http://www.uml.org/) is a trademark of Object Management Group (OMG).
**Program Objects**

An OpenCL program consists of a set of kernels that are identified as functions declared with the `__kernel` qualifier in the program source.

**Create Program Objects**

- `cl_program clCreateProgramWithSource (cl_context context, const void *source, size_t size, const char *allowlisted_kernels, const char *sourcelist, const cl_device_id *device_list, void *user_data)`
- `cl_program clCreateProgramWithBinary (cl_context context, const void *binary, size_t size, const char *binary_name, const cl_device_id *device_list, void *user_data)`
- `cl_program clCreateProgramWithBuiltInKernels (cl_context context, const char *kernels, const cl_device_id *device_list, void *user_data)`

**Enqueuing SVM Operations**

- `cl_int clEnqueueSVMFree (cl_context context, void *svm_pointer)`
- `cl_int clEnqueueSVMMap (cl_command_queue command_queue, cl_uint num_svm_pointers, const char *svm_names, size_t *svm_sizes, const void *svm_pointers, const cl_device_id *device_list, const void *user_data)`
- `cl_int clEnqueueSVMMemcpy (cl_command_queue command_queue, void *dst_data, const char *src_data, size_t size, const cl_device_id *device_list, const void *user_data)`

**Query Program Objects**

- `cl_int clGetProgramInfo (cl_program program, cl_program_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`
- `cl_int clGetProgramBuildInfo (cl_program program, cl_device_id device_id, cl_command_queue command_queue, cl_int error_code, void *user_data)`

**Query Kernel Information**

- `cl_int clGetKernelInfo (cl_kernel kernel, cl_int error_code, void *user_data)`

**Query Kernel Argument Information**

- `cl_int clGetKernelArgInfo (cl_kernel kernel, size_t *parameter_size, void *parameter_value, const void *svm_pointer)`

**SVM Sharing Granularity**

See more on SVM on page 4 of this reference guide.

**Kernel Objects**

A kernel object encapsulates the specific `__kernel` function and the arguments to be used when executing it. Items in blue apply when the appropriate extension is supported.

**Create Kernel Objects**

- `cl_kernel clCreateKernel (cl_program program, const char *kernel_name, const cl_int error_code)`
- `cl_kernel clCreateKernelsInProgram (cl_program program, size_t num_kernels, const cl_int kernels, const cl_int num_kernels_ret)`
- `cl_int clRetainKernel (cl_kernel kernel)`
- `cl_int clReleaseKernel (cl_kernel kernel)`

**Kernel Arguments and Queries**

- `cl_int clSetKernelArgSVMPointer (cl_kernel kernel, const cl_int arg_index, void *svm_pointer)`
- `cl_int clSetKernelArgExtInfo (cl_kernel kernel, cl_int error_code, void *user_data)`
- `cl_int clGetKernelInfo (cl_kernel kernel, cl_int error_code, void *user_data)`

**Flush and Finish**

- `cl_int clFlush (cl_command_queue command_queue)`
- `cl_int clFinish (cl_command_queue command_queue)`

---

**Optimization options**

- `-cl-opt-disable` to disable all optimizations
- `-cl-enable` to enable all optimizations
- `-cl-no-signed-zeros` to use signed zeros
- `-cl-finite-math-only` to use finite math
- `-cl-fast-relaxed-math` to use fast relaxed math
- `-cl-uniform-work-group-size` to specify uniform work group size

**Warning request/suppression**

- `-w` to suppress warnings

**Control OpenCL language version**

- `-cl-version CL_1.1` to specify OpenCL 1.1 language
- `-cl-version CL_1.2` to specify OpenCL 1.2 language
- `-cl-version CL_2.0` to specify OpenCL 2.0 language

**Library linking options**

- `-create-library` to create a library
- `-enable-link-options` to enable link options

**Program linking options**

- `-cl-denorms-are-zero` to enable denorms are zero
- `-cl-finite-only` to enable finite only
- `-cl-fast-relaxed-math` to enable fast relaxed math
- `-cl-unsafe-math-optimizations` to enable unsafe math optimizations
### Summary of SVM Options in OpenCL

<table>
<thead>
<tr>
<th>SVM</th>
<th>Granularity of sharing</th>
<th>Memory allocation</th>
<th>Mechanisms to enforce consistency</th>
<th>Explicit updates between host and device?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-SVM buffers</td>
<td>OpenCL Memory objects (buffer)</td>
<td>clCreateBuffer</td>
<td>Host synchronization points on the same or between devices.</td>
<td>Yes, through Map and Unmap commands.</td>
</tr>
<tr>
<td>Coarse-Grained buffer SVM</td>
<td>OpenCL Memory objects (buffer)</td>
<td>clSVMAlloc</td>
<td>Host synchronization points between devices</td>
<td>Yes, through Map and Unmap commands.</td>
</tr>
<tr>
<td>Fine Grained buffer SVM</td>
<td>Bytes within OpenCL Memory objects (buffer)</td>
<td>clSVMAlloc</td>
<td>Synchronization points plus atomics (if supported)</td>
<td>No</td>
</tr>
<tr>
<td>Fine-Grained system SVM</td>
<td>Bytes within Host memory (system)</td>
<td>Host memory allocation mechanisms (e.g. malloc)</td>
<td>Synchronization points plus atomics (if supported)</td>
<td>No</td>
</tr>
</tbody>
</table>

### Memory Model: Shared Virtual Memory [3.3.3]

OpenCL extends the global memory region into the host memory region through a shared virtual memory (SVM) mechanism. There are three types of SVM in OpenCL:

- **Coarse-Grained buffer SVM**: Sharing occurs at the granularity of regions of OpenCL buffer memory objects. Consistency is enforced at synchronization points and with map/unmap commands to drive updates between the host and the device. This form of SVM is similar to the use of cl_mem buffers, with two differences. First, it lets kernel-instances share pointer-based data structures (such as linked-lists) with the host program. Second, concurrent access by multiple kernels on the same device is valid as long as the set of concurrently executing kernels is bounded by synchronization points. Concurrent access by multiple kernels on the same device is valid as long as the set of kernels is bounded by synchronization points. This form of SVM is similar to non-SVM use of memory; however, it lets kernel-instances share pointer-based data structures (such as linked-lists) with the host program. Program scope global variables are treated as per-device coarse-grained SVM for addressing and sharing purposes.

- **Fine-Grained buffer SVM**: Sharing occurs at the granularity of individual loads/stores into bytes within OpenCL buffer memory objects. Loads and stores may be cached. This means consistency is guaranteed at synchronization points. If the optional OpenCL atomics are supported, they can be used to provide fine-grained control of memory consistency.

- **Fine-Grained system SVM**: Sharing occurs at the granularity of individual loads/stores into bytes occurring anywhere within the host memory. Loads and stores may be cached so consistency is guaranteed at synchronization points. If the optional OpenCL atomics are supported, they can be used to provide fine-grained control of memory consistency.

Coarse-Grained buffer SVM is required in the core OpenCL specification. The two finer-grained approaches are optional features in OpenCL. The various SVM mechanisms to access host memory from the work-items associated with a kernel instance are summarized in table 3-2 below.
Supported Data Types
The opencl double scalar and vector types are supported if cl_device_double_fp_config is not zero.

Built-in Scalar Data Types [6.1.1]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td></td>
<td>true (1) or false (0)</td>
</tr>
<tr>
<td>char</td>
<td>cl_char</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>unsigned char, uchar</td>
<td>cl_uchar</td>
<td>8-bit unsigned</td>
</tr>
<tr>
<td>short</td>
<td>cl_short</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>unsigned short, ushort</td>
<td>cl_ushort</td>
<td>16-bit unsigned</td>
</tr>
<tr>
<td>int</td>
<td>cl_int</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>unsigned int, uint</td>
<td>cl_uint</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>long</td>
<td>cl_long</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>unsigned long, ulong</td>
<td>cl_ulong</td>
<td>64-bit unsigned</td>
</tr>
<tr>
<td>float</td>
<td>cl_float</td>
<td>32-bit float</td>
</tr>
<tr>
<td>double</td>
<td>cl_double</td>
<td>64-bit IEEE 754</td>
</tr>
<tr>
<td>half</td>
<td>cl_half</td>
<td>16-bit float (storage only)</td>
</tr>
<tr>
<td>size_t</td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>ptrdiff_t</td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>intptr_t</td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>uintptr_t</td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>void</td>
<td></td>
<td>void</td>
</tr>
</tbody>
</table>

Built-in Vector Data Types [6.1.2]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float4</td>
<td></td>
<td>4-component vector</td>
</tr>
<tr>
<td>float8</td>
<td></td>
<td>8-component vector</td>
</tr>
<tr>
<td>float16</td>
<td></td>
<td>16-component vector</td>
</tr>
<tr>
<td>float32</td>
<td></td>
<td>32-component vector</td>
</tr>
<tr>
<td>float64</td>
<td></td>
<td>64-component vector</td>
</tr>
</tbody>
</table>

Reserved Data Types [6.1.4]

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>boolean vector</td>
</tr>
<tr>
<td>half</td>
<td>16-bit vector</td>
</tr>
<tr>
<td>quad, quadn</td>
<td>128-bit float vector</td>
</tr>
<tr>
<td>complex</td>
<td>16-bit complex</td>
</tr>
<tr>
<td>complex float</td>
<td>complex float</td>
</tr>
<tr>
<td>complex double, complex doublen</td>
<td>complex double, complex doublen</td>
</tr>
<tr>
<td>complex quad, complex quadn</td>
<td>complex quad, complex quadn</td>
</tr>
<tr>
<td>float*</td>
<td>n*m matrix of 32-bit floats</td>
</tr>
<tr>
<td>double*</td>
<td>n*m matrix of 64-bit floats</td>
</tr>
</tbody>
</table>

Vector Component Addressing [6.1.7]

Vector Components

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>float2 v;</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float3 v;</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vz, vs2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float4 v;</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vz, vs2</td>
<td>vw, vs3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float8 v;</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vz, vs2</td>
<td>vw, vs3</td>
<td>vs4</td>
<td>vs5</td>
<td>vs6</td>
<td>vs7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float16 v;</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vz, vs2</td>
<td>vw, vs3</td>
<td>vs4</td>
<td>vs5</td>
<td>vs6</td>
<td>vs7</td>
<td>vs8</td>
<td>vs9</td>
<td>vs10</td>
<td>vs11</td>
<td>vs12</td>
<td>vs13</td>
<td>vs14</td>
<td>vs15</td>
</tr>
</tbody>
</table>

Vector Addressing Equivalences
Numeric indices are preceded by the letter s or S, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: vx, vxx, vlo.

<table>
<thead>
<tr>
<th></th>
<th>vlo</th>
<th>vhi</th>
<th>uodd</th>
<th>ueven</th>
</tr>
</thead>
<tbody>
<tr>
<td>float2</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vx, vs0</td>
<td></td>
</tr>
<tr>
<td>float3</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vz, vs2</td>
<td></td>
</tr>
<tr>
<td>float4</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vz, vs2</td>
<td>vw, vs3</td>
</tr>
<tr>
<td>float8</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vz, vs2</td>
<td>vw, vs3</td>
</tr>
<tr>
<td>float16</td>
<td>vx, vs0</td>
<td>vy, vs1</td>
<td>vz, vs2</td>
<td>vw, vs3</td>
</tr>
</tbody>
</table>

Operators and Qualifiers

Operators [6.3]
These operators behave similarly as in C99 except operands may include vector types when possible:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>*</td>
</tr>
<tr>
<td>++</td>
<td>=</td>
</tr>
<tr>
<td>&gt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>op=</td>
</tr>
</tbody>
</table>

Address Space Qualifiers [6.5]

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_global</td>
<td>global</td>
</tr>
<tr>
<td>__local</td>
<td>local</td>
</tr>
<tr>
<td>__constant</td>
<td>constant</td>
</tr>
<tr>
<td><strong>attribute</strong>((private))</td>
<td>private</td>
</tr>
</tbody>
</table>

Function Qualifiers [6.7]

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__kernel</td>
<td>kernel</td>
</tr>
<tr>
<td><strong>attribute</strong>((aligned))</td>
<td>attribute _ (aligned)</td>
</tr>
<tr>
<td><strong>attribute</strong>((packed))</td>
<td>attribute _ (packed)</td>
</tr>
<tr>
<td><strong>attribute</strong>((noalign))</td>
<td>attribute _ (noalign)</td>
</tr>
<tr>
<td><strong>attribute</strong>((noscm))</td>
<td>attribute _ (noscm)</td>
</tr>
<tr>
<td><strong>attribute</strong>((__noalign))</td>
<td>attribute _ (noalign)</td>
</tr>
<tr>
<td><strong>attribute</strong>((__noscm))</td>
<td>attribute _ (noscm)</td>
</tr>
</tbody>
</table>

Attribute Qualifiers [6.11]
Use to specify attributes of enum, struct, and union types:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong>((aligned))</td>
<td>attribute _ (aligned)</td>
</tr>
<tr>
<td><strong>attribute</strong>((packed))</td>
<td>attribute _ (packed)</td>
</tr>
<tr>
<td><strong>attribute</strong>((noalign))</td>
<td>attribute _ (noalign)</td>
</tr>
<tr>
<td><strong>attribute</strong>((noalign))</td>
<td>attribute _ (noalign)</td>
</tr>
</tbody>
</table>

Use to specify attributes of variables or structure fields:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong>((aligned))</td>
<td>attribute _ (aligned)</td>
</tr>
<tr>
<td><strong>attribute</strong>((alignof))</td>
<td>attribute _ (alignof)</td>
</tr>
</tbody>
</table>

Use to specify basic blocks and control-flow statements:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong>((atomic))</td>
<td>attribute _ (atomic)</td>
</tr>
</tbody>
</table>

Use to specify that a loop (for, while, and do loops) can be unrolled. (Must appear immediately before the loop to be affected.)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong>((opencl_unroll))</td>
<td>attribute _ (opencl_unroll)</td>
</tr>
</tbody>
</table>

Preprocessor Directives & Macros [6.10]

#pragma OPENCL FP_CONTRACT on-off-switch: ON, OFF, DEFAULT

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>FILE</em></td>
<td>current source file</td>
</tr>
<tr>
<td><em>func</em></td>
<td>current function name</td>
</tr>
<tr>
<td><em>LINE</em></td>
<td>integer line number</td>
</tr>
<tr>
<td>_<em>OPENCL_VERSION</em></td>
<td>integer version number, e.g: 200</td>
</tr>
<tr>
<td>_<em>CL_VERSION_1_0</em></td>
<td>substitutes integer 100 for 1.0</td>
</tr>
<tr>
<td>_<em>CL_VERSION_1_1</em></td>
<td>substitutes integer 110 for 1.1</td>
</tr>
<tr>
<td>_<em>CL_VERSION_1_2</em></td>
<td>substitutes integer 120 for 1.2</td>
</tr>
<tr>
<td>_<em>CL_VERSION_2_0</em></td>
<td>substitutes integer 200 for 2.0</td>
</tr>
<tr>
<td>_<em>OPENCL_C_VERSION</em></td>
<td>sub integer for OpenCL C version</td>
</tr>
<tr>
<td>_<em>ENDIAN_LITTLE</em></td>
<td>1 if device is little endian</td>
</tr>
<tr>
<td>_<em>IMAGE_SUPPORT</em></td>
<td>1 if images are supported</td>
</tr>
<tr>
<td>_<em>FAST_RELAXED_MATH</em></td>
<td>1 if -O2-fast-relax math optimization option is specified</td>
</tr>
<tr>
<td>_<em>FP_FMA</em></td>
<td>defined if double fma is fast</td>
</tr>
<tr>
<td>_<em>FP_FMAF</em></td>
<td>defined if float fma is fast</td>
</tr>
<tr>
<td>_<em>FP_FMA_HALF</em></td>
<td>defined if half fma is fast</td>
</tr>
<tr>
<td>__kernel_exec(X, ytype)</td>
<td>same as:</td>
</tr>
</tbody>
</table>

©2015 Khronos Group - Rev. 1115 www.khronos.org/opencl
Blocks [6.12]
A result value type with a list of parameter types, similar to a function type. In this example:
1. The * declares variable "myBlock" is a Block.
2. The return type for the Block "myBlock" is int.
3. myBlock takes a single argument of type float.
4. The argument is named "y.
5. Multiplier captured from block’s environment.

```c
int ("myBlock") (int) = "(int num) {return num * multiplier; }
```

Math Built-in Functions [6.13.2] [9.4.2]
T is type float, optionally double, or half if the cl_khr_fp16 extension is enabled. Tn is the vector form of T, where n is 2, 3, 4, 8, or 16. T is Ts and Tn. All angles are in radians.

HN indicates that half and native variants are available using only the float or float variants by prepending "half_" or "native_," to the function name. Prototypes shown in brown text are available in half_ and native_ forms only using only the float or float

```c
fmin (T x, y) Return y if y < x, otherwise it returns x
```

```
round (T x) Integral value nearest to x rounding
```

Work-Item Built-in Functions [6.13.1]
Query the number of dimensions, global, and local work size specified to clEnqueueNDRangeKernel, and global and local identifier of each work-item when this kernel is executed on a device. Sub-groups require the cl_khr_subgroups extension.

```c
uint get_work_dim () Number of dimensions in use
```

```
size_t get_global_id (uint dim) Number of global work-items
```

```
size_t get_global_id (uint dim) Global work-item ID value
```

```
size_t get_local_id (uint dim) Number of local work-items if kernel executed with uniform work-group size
```

```
size_t get_local_size (uint dim) Number of local work-items
```

```
size_t get_max_local_size (uint dim) Maximum size of a work-group
```

```
size_t get_global_linear_id () Work-items 1-dimensional global ID
```

```
size_t get_local_linear_id () Work-items 1-dimensional local ID
```

```
size_t get_sub_group_id () Number of work-items in the subgroup
```

```
size_t get_num_groups (uint dim) Number of work-groups
```

```
size_t get_sub_group_id (uint dim) Sub-group ID
```

```
size_t get_sub_group_local_id (uint dim) Unique work-item ID
```

Math Constants [6.13.2] [9.4.2]
The values of the following symbolic constants are single-precision float.

```
M_E_F Value of e
```

```
M_LOG2E_F Value of log2 e
```

```
M_LOG10E_F Value of log10 e
```

```
M_LN2_F Value of log2 e
```

```
M_LN10_F Value of log10 e
```

```
M_PI_F Value of π
```

```
M_PI_2_F Value of π/2
```

```
M_PI_4_F Value of π/4
```

```
M_SQRT1_2_F Value of 2/√π
```

```
M_SQRT1_2_ar F Value of 2/π
```

When double precision is supported, macros ending in _F are available in type double by removing _F from the macro name, and in type half when the cl_khr_fp16 extension is enabled by replacing _F with _HN.

©2015 Khronos Group - Rev. 1115
www.khronos.org/opencl
### Integer Built-in Functions [6.13.3]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T_sub_sat(T x, T y)</code></td>
<td><code>x - y</code> and saturates the result</td>
</tr>
<tr>
<td><code>T_popcount(T x)</code></td>
<td>Non-zero bits in <code>x</code></td>
</tr>
<tr>
<td><code>T_abs(T x)</code></td>
<td><code>x</code></td>
</tr>
<tr>
<td><code>T_abs_diff(T x, T y)</code></td>
<td><code>x - y</code> without modulo overflow</td>
</tr>
<tr>
<td><code>T_add_sat(T x, T y)</code></td>
<td><code>x + y</code> and saturates the result</td>
</tr>
<tr>
<td><code>T_hadd(T x, T y)</code></td>
<td><code>(x + y) &gt; 1</code> without modulo overflow</td>
</tr>
<tr>
<td><code>T_radd(T x, T y)</code></td>
<td><code>(x + y) = 1</code></td>
</tr>
<tr>
<td><code>T_cmstock(T x, T y)</code></td>
<td>Number of leading 0-bits in <code>x</code></td>
</tr>
<tr>
<td><code>T_cz(T x)</code></td>
<td>Number of trailing 0-bits in <code>x</code></td>
</tr>
<tr>
<td><code>T_max(T x, T y)</code></td>
<td><code>max(x, y)</code></td>
</tr>
<tr>
<td><code>T_max(T x, T y)</code></td>
<td><code>max(T x, T y)</code></td>
</tr>
<tr>
<td><code>T_min(T x, T y)</code></td>
<td><code>min(x, y)</code></td>
</tr>
<tr>
<td><code>T_min(T x, T y)</code></td>
<td><code>min(T x, T y)</code></td>
</tr>
<tr>
<td><code>T_mul_hi(T a, T b, T c)</code></td>
<td><code>a * b</code> and saturates the result</td>
</tr>
<tr>
<td><code>T_rotate(T v, T x)</code></td>
<td>`result[0] = x[0] &lt;&lt; 16</td>
</tr>
</tbody>
</table>

#### Relational Built-in Functions [6.13.4]

These functions can be used with built-in scalar or vector types as arguments and return a scalar or vector integer result. `T` is type float, floatT, char, charT, uchar, ucharT, short, shortT, shortD, shortT, ushort, ushortT, int, intT, uint, uintT, long, longT, longD, longT, ulong, ulongT, or optionally double or doubleT. `T` is type char, charT, short, shortT, shortD, shortT, ushort, ushortT, int, intT, uint, uintT, long, longT, longD, longT, ulong, ulongT, or optionally double or doubleT.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int isequal(float x, float y)</code></td>
<td>Compare of <code>x == y</code></td>
</tr>
<tr>
<td><code>int inotequal(float x, float y)</code></td>
<td>Compare of <code>x != y</code></td>
</tr>
<tr>
<td><code>int isless(double x, double y)</code></td>
<td>Compare of <code>x &lt; y</code></td>
</tr>
<tr>
<td><code>int islessor_equal(double x, double y)</code></td>
<td>Compare of `(x &lt; y)</td>
</tr>
<tr>
<td><code>int isgreater(double x, double y)</code></td>
<td>Compare of <code>x &gt; y</code></td>
</tr>
<tr>
<td><code>int isgreateror_equal(double x, double y)</code></td>
<td>Compare of `(x &gt; y)</td>
</tr>
<tr>
<td><code>int isfinite(float x)</code></td>
<td>Test for finite value</td>
</tr>
<tr>
<td><code>int isnan(float x)</code></td>
<td>Test for a NaN</td>
</tr>
<tr>
<td><code>int isnormal(float x)</code></td>
<td>Test for a normal value</td>
</tr>
<tr>
<td><code>int isunordered(float x)</code></td>
<td>Test if arguments are unordered</td>
</tr>
<tr>
<td><code>int isinf(float x)</code></td>
<td>Test for infinity</td>
</tr>
<tr>
<td><code>int isnan(float x)</code></td>
<td>Test for a NaN</td>
</tr>
<tr>
<td><code>int isnormal(float x)</code></td>
<td>Test for a normal value</td>
</tr>
<tr>
<td><code>int isunordered(float x)</code></td>
<td>Test if arguments are unordered</td>
</tr>
<tr>
<td><code>int islessequal(double x, double y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(double x, double y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(float x, float y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(float x, float y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(double x, double y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(double x, double y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(float x, float y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(float x, float y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(double x, double y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(double x, double y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(float x, float y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(float x, float y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(double x, double y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(double x, double y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(float x, float y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(float x, float y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(double x, double y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(double x, double y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(float x, float y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(float x, float y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(double x, double y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(double x, double y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(float x, float y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(float x, float y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(double x, double y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(double x, double y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
<tr>
<td><code>int islessequal(float x, float y)</code></td>
<td>Compare of <code>x &lt;= y</code></td>
</tr>
<tr>
<td><code>int isgreater_equal(float x, float y)</code></td>
<td>Compare of `(x &lt;= y)</td>
</tr>
</tbody>
</table>

### Geometric Built-in Functions [6.13.5][9.4.4]

These functions operate component-wise and use round to nearest even rounding mode. `T` is type float, optionally double, or half if the half extension is enabled. `T` is `T` and the `2`, `3`, or `4`-component vector forms of `T`

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>float fast_distance(float p0, float p1)</code></td>
<td>Vector distance</td>
</tr>
<tr>
<td><code>float fast_distance(float p0, float p1)</code></td>
<td>Vector distance</td>
</tr>
<tr>
<td><code>float fast_length(float p)</code></td>
<td>Vector length</td>
</tr>
<tr>
<td><code>float fast_length(float p)</code></td>
<td>Vector length</td>
</tr>
<tr>
<td><code>float fast_normalize(float p)</code></td>
<td>Normal vector length</td>
</tr>
<tr>
<td><code>float fast_normalize(float p)</code></td>
<td>Normal vector length</td>
</tr>
</tbody>
</table>

### Common Built-in Functions [6.13.4][9.4.3]

These functions operate component-wise and use round to nearest even rounding mode. `T` is type float, optionally double, or half if the half extension is enabled. `Tn` is the vector form of `T`, where `n` is `2`, `3`, or `4`, or `16`. `T` is `T` and `Tn`.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T_clamp(T x, T min, T max)</code></td>
<td>Clamp <code>x</code> to range given by min, max</td>
</tr>
<tr>
<td><code>T_degrees(T radians)</code></td>
<td>Radians to degrees</td>
</tr>
<tr>
<td><code>T_max(T x, T y)</code></td>
<td>Mix of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>T_min(T x, T y)</code></td>
<td>Min of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>T_mix(T x, T y, T a)</code></td>
<td>Linear blend of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>Tueue(T r, T edge, T x)</code></td>
<td>Step and interpolate</td>
</tr>
<tr>
<td><code>T_sign(T x)</code></td>
<td>Sign of <code>x</code></td>
</tr>
</tbody>
</table>

©2015 Khronos Group - Rev. 1115

www.khronos.org/opencl
## OpenCL 2.1 Reference Card

### Vector Data Load/Store [6.13.7] [9.4.6]

- `memory_order_relaxed`  
- `memory_order_acquire`  
- `memory_order_release`

<table>
<thead>
<tr>
<th>In <code>vloadn</code> (size_t offset, const [constant] *p)</th>
<th>Read vector data from address (p + offset * n)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vstoren</code> (In, size_t offset, const [constant] half *p)</td>
<td>Write vector data to address (p + offset * n)</td>
</tr>
<tr>
<td><code>float vloadl_half</code> (size_t offset, const [constant] half *p)</td>
<td>Read a half from address (p + offset)</td>
</tr>
<tr>
<td><code>float vloada_half</code> (double * data, size_t offset, half *p)</td>
<td>Read a half from address (p + offset * n)</td>
</tr>
</tbody>
</table>

### Synchronization & Memory Fence Functions [6.13.8]

Flags argument is the memory address space, set to 0 or an OR’d combination of `CLK_X_MEM_FENCE` where X may be `LOCAL`, `GLOBAL`, or `IMAGE`. Memory fence functions provide ordering between memory operations of a work-item. Sub-groups require the `cl_khr_subgroups` extension.

- `void work_group_barrier(cl_mem_fence_flags memory_scope)`
  - Work-items in a work-group must execute this before any can continue
- `void atomic_work_item_fence(cl_mem_fence_flags memory_scope)`
  - Orders loads and stores of a work-item executing a kernel
- `void sub_group_barrier(cl_mem_fence_flags memory_scope)`
  - Work-items in a sub-group must execute this before any can continue

### Atomic Functions [6.13.11]

OpenCL implements a subset of the C11 atomics (see section 7.17 of the C11 specification) and synchronization operations. In the following tables, A refers to an atomic_* type (not including atomic_flag). C refers to its corresponding non-atomic type. M refers to the type of the other argument for arithmetic operations. For atomic integer types, M is C. For atomic pointer types, M is `void`.

- The type atomic_* is a 32-bit integer. `atomic_long` and `atomic_ulong` require extension `cl_khr_int64_base_atomics` or `cl_khr_int64_base_atomics_and_sync`
- The atomic_double type requires double precision support. The default scope is `work_group` for local atomics and all `svm`_devices for global atomics. The extensions `cl_khr_int64_base_atomics` and `cl_khr_int64_base_atomics_and_sync` implement atomic operations on 64-bit signed and unsigned integers to locations in `__global` and `__local` memory.
- See the table under Atomic Types and Enum Constants for information about parameter types, memory_scope, and memory_flag.

<table>
<thead>
<tr>
<th>void atomic_incinit volatile A *obj, C value</th>
<th>Initializes the atomic object pointed to by obj to the value value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>void atomic_work_item_fence(cl_mem_fence_flags memory_order, event_t *p)</td>
<td>Effects based on value of order: flags must be <code>CLK_GLOBAL, LOCAL, IMAGE_MEM_FENCE</code> or a combination of these.</td>
</tr>
<tr>
<td>void atomic_store(store volatile A *object, C desired)</td>
<td>Atomically replace the value pointed to by object with the value of desired. Memory is affected according to the value of order.</td>
</tr>
<tr>
<td>C atomic_loadload volatile A *object, memory_order order</td>
<td>Atomically returns the value pointed to by object. Memory is affected according to the value of order.</td>
</tr>
<tr>
<td>C atomic_exchange (volatile A *object, C desired)</td>
<td>Atomically replaces the value pointed to by object with desired. Memory is affected according to the value of order.</td>
</tr>
<tr>
<td>bool atomic_compare_exchange_int64 (volatile A *object, C expected, C desired)</td>
<td>Atomically compares the value pointed to by object for equality with that in expected, and if true, replaces the value pointed to by object with desired, and if false, updates the value in expected with the value pointed to by object. These operations are atomic read-modify-write operations.</td>
</tr>
<tr>
<td>bool atomic_fetch_and_op (volatile A *object, M operand)</td>
<td>Atomically replaces the value pointed to by object with the result of the computation applied to the value pointed to by object and the given operand.</td>
</tr>
</tbody>
</table>

### Async Copies and Prefetch [6.13.10] [9.4.7]

1. `atomic_flag test_and_set (volatile A *object)`: Atomically sets the value pointed to by object to true. Memory is affected according to the value of order. Returns atomically, the value of the object immediately before the effects.
2. `atomic_flag clear (volatile A *object)`: Atomically sets the value pointed to by object to false. The order argument shall not be `memory_order_acquire` or `memory_order_acquire_rel`. Memory is affected according to the value of order.

### Values for key for atomic_fetch and modify operations

<table>
<thead>
<tr>
<th>key</th>
<th>op</th>
<th>computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>sub</td>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td>bitwise inclusive or</td>
</tr>
<tr>
<td>xor</td>
<td>^</td>
<td>bitwise exclusive or</td>
</tr>
</tbody>
</table>

### Atomic Types and Enum Constants

- `memory_scope_sub_group` requires the `cl_khr_subgroups` extension.
- `memory_order` requires the `cl_khr_subgroups` and `cl_khr_int64_base_atomics` extensions.

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>memory_order</code></td>
<td><code>memory_order_relaxed</code> <code>memory_order_acquire</code> <code>memory_order_release</code></td>
</tr>
<tr>
<td><code>memory_order_acq_rel</code></td>
<td><code>memory_order_acquire</code> <code>memory_order_release</code></td>
</tr>
<tr>
<td><code>memory_order_sea_cst</code></td>
<td><code>memory_order_sea_cst</code></td>
</tr>
<tr>
<td><code>memory_scope</code></td>
<td><code>memory_scope_work_item</code> <code>memory_scope_work_group</code> <code>memory_scope_sub_group</code></td>
</tr>
<tr>
<td><code>memory_scope_device</code> (default for functions that don't take a <code>memory_scope</code> argument)</td>
<td></td>
</tr>
</tbody>
</table>

### Atomic Integer and floating-point types

- Indicates types supported by a limited subset of atomic operations
- Indicates sizes depends on whether implemented on 64-bit or 32-bit architecture
- Indicates types supported only if both 64-bit extensions are supported.

- `atomic_int` `atomic_int_t`
- `atomic_uint` `atomic_uint_t`
- `atomic_long` `atomic_long_t`
- `atomic_ulong` `atomic_ulong_t`
- `atomic_double` `atomic_double_t`
- `atomic_intptr_t` `atomic_intptr_t_t`
- `atomic_size_t` `atomic_size_t_t` `atomic_ptrdiff_t` `atomic_ptrdiff_t_t`

### Atomic Macros

- `#define ATOMIC_VAR_INIT(C value)`
- `#define ATOMIC_FLAG_INIT`  
- Expands to a token sequence to initialize an atomic object of a type that is initialization-compatible with value.
- `#define ATOMIC_FLAG_INIT`  
- Initializes an atomic flag to the clear state.
**Address Space Qualifier Functions** [6.13.9]

T refers to any of the built-in data types supported by OpenCL C or a user-defined type.

- `[const] global T * _global_ (const T *ptr)`
  - global address space
- `[const] local T * _local_ (const T *ptr)`
  - local address space
- `[const] private T * _private_ (const T *ptr)`
  - private address space
- `[const] cl_mem_fence_flags get_fence((const T *ptr))`  
  - Memory fence values: `CLK_GLOBAL_MEM_FENCE`, `CLK_LOCAL_MEM_FENCE`  

**Workgroup Functions** [6.13.15] [9.17.3.4]  

T is type int, uint, long, ulong, or float, optionally double, or half if the cl_khr_fp16 extension is supported. Subgroups require the `cl_khr_subgroups` extension. Double and vector types require double precision support.

- `int work_group_all(int predicate)`
- `int work_group_any(int predicate)`
- `int work_group_all(uint predicate)`
- `int work_group_any(uint predicate)`

Returns a non-zero value if predicate evaluates to non-zero for all or any workitems in the workgroup or sub-group.

**print Function** [6.13.13]  

- `void print(const char * restrict format, ...)`

**print output synchronization**  

When the event associated with a particular kernel invocation completes, the output of applicable print calls is flushed to the implementation-defined output stream.

**printf format string**  

- `char %flags[width][precision][vector][length] conversion`

**Examples:**

- The following examples show the use of the vector specifier in the printf format string.
  - `uchar4 uc = (uchar4)(0xFA, 0xFB, 0xFC, 0xFD);`
  - `float4  f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);`
  - `uchar uc = (uchar)uc;`

**Examples:**

- `float4 f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);`
  - `printf("%2.2v4f\n", f);`
  - `Output:  f4 = 1.00,2.00,3.00,4.00`

**Pipe Built-in Functions** [6.13.16-2.4]

T represents the built-in OpenCL C scalar or vector integer or floating-point data types or any user defined type built from these scalar and vector data types. Half scalar and vector types require the `cl_khr_fp16` extension. Subgroups require the `cl_khr_subgroups` extension. Double or vector double types require double precision support.

- `int read_pipe(_read_only pipe T p, T *ptr)`
  - Read packet from p into ptr.
- `int read_pipe(_read_only pipe T p, T *ptr)`
  - Read packet from reserved area of the pipe & index into ptr.
- `int write_pipe(_write_only pipe T p, T *ptr)`
  - Write packet specified by ptr to p.
- `int write_pipe(_write_only pipe T p, T *ptr)`
  - Write packet specified by ptr to reserved area & index.

**Miscellaneous Vector Functions** [6.13.12]

Tm and Tn are type char, uchar, short, ushort, int, uint, long, ulong, float, optionally double, or half if the cl_khr_fp16 extension is supported, where m is 2, 4, 8, or 16 except in vec2 it may be 3. Tnm is uchar, ushort, uint, or ulong.

- `int vec_step(Tn d)`
- `int vec_step(Typename)`
  - Takes built-in scalar or vector data type argument. Returns 1 for scalar, 4 for 3-component vector, else number of elements in the specified type.
- `Tm shuffle(Tm x, TUn mask)`
  - Construct permutation of elements from one or two input vectors, return a vector with same element type as input and length that is the same as the shuffle mask.

**Enqueuing and Kernel Query Built-in Functions** [6.13.17] [9.17.3.6]

A kernel may enqueue code represented by Block syntax, and control execution order with event dependencies including user events and markers. There are several advantages to using the Block syntax: it is more compact; it does not require a cl_kernel object; and enqueuing can be done as a single semantic step. Subgroups require the cl_khr_subgroups extension. The macro `CLK_NULL_EVENT` refers to an invalid device event. The macro `CLK_NULL_QUEUE` refers to an invalid device queue.

- `int enqueue_kernel(queue_t queue, kernel enqueue flags T flags, const ndrange_t ndrange, void *(block)void)`
  - Allows a work-item to enqueue a block for execution to queue. Work-items can enqueue multiple blocks to a device queue(s).
  - flags may be one of `CLK_ENQUEUE_FLAGS`.  
    - `ND_RANGE_WAIT_ALL`  
    - `ND_RANGE_WAIT_KERNEL_WAIT_WORKGROUP`  

- `uint get_kernel_work_group_size(void *(block)void)`
  - Query the maximum work-group size that can be used to execute a block.
- `uint get_kernel_work_group_size(void *(block)local void *, ...)`
  - Returns the preferred multiple of work-group size for launch.
- `uint get_kernel_preferred_work_group_size_multiple(void *(block)local void *)`
  - Returns the preferred multiple of work-group size for launch.
- `uint get_kernel_preferred_work_group_size_multiple(void *(block)local void *, ...)`
  - Returns the preferred multiple of work-group size for launch.
- `int enqueue_marker(queue_t queue, uint num_events_in_wait_list, const clk_event_t *event_wait_list, clk_event_t *event_ret)`
  - Enqueue a marker command to queue.
- `int enqueue_sub_group_count_for_ndrange(const ndrange_t ndrange, void *(block)void)`
  - Returns number of subgroups in each工作组的 dispatch.
- `int enqueue_sub_group_count_for_ndrange(const ndrange_t ndrange, void *(block)void *, ...)`
  - Returns the maximum sub-group size for a block.
**Event Built-in Functions**

T is type int, uint, long, ulong, or float, optionally double, or half if the cl_khr_fp16 extension is enabled.

- **void retain_event(cl_event_t event)** increments event reference count.
- **void release_event(cl_event_t event)** decrements event reference count.
- **void clk_event_t create_user_event()** creates a user event.
- **bool is_valid_event(cl_event_t event)** returns true for valid event.
- **void set_user_event_status(cl_event_t event, int status)** sets the execution status of a user event. status: CL_COMPLETE or a negative error value.
- **void capture_event profiling_info(cl_event_t event, cl_profiling_info_name, global void *value)** captures profiling information for command associated with event in value.

**Helper Built-in Functions**

- **cl_int clEnqueueWriteImage(cl_command_queue command_queue, cl_mem image, cl_mem dst_buffer, cl_mem image_info, cl_bool blocking_read, const size_t *origin, const size_t *region, size_t input_row_pitch, size_t input_slice_pitch, cl_int num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)***
- **cl_int clEnqueueFillImage(cl_command_queue command_queue, cl_mem image, void *fill_color, const size_t *origin, const size_t *region, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)***
- **cl_int clEnqueueCopyImage(cl_command_queue command_queue, cl_mem src_image, cl_mem dst_image, const size_t *src_origin, const size_t *dst_origin, const size_t *region, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)***
- **cl_int clEnqueueCopyImageToBuffer(cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer, cl_mem image_info, cl_bool blocking_read, const size_t *origin, const size_t *region, size_t src_image_row_pitch, size_t src_image_slice_pitch, cl_int num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)***
- **cl_int clEnqueueCopyBufferToImage(cl_command_queue command_queue, cl_mem src_buffer, cl_mem dst_image, size_t src_offset, const size_t *dst_origin, const size_t *region, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)***

**OpenCL Image Processing Reference**

A subset of the OpenCL API 2.1 and C Language 2.0 specifications pertaining to image processing and graphics.

**Image Formats**

Supported image formats: image_channel_order with image_channel_data_type.

**Image Channel Data Types**

- CL_A: CL_HALF_FLOAT, CL_FLOAT,
- CL_LUMINANCE: CL_SIGNED, CL_UNSIGNED, CL_HALF_FLOAT, CL_FLOAT,
- CL_A1: CL_HALF_FLOAT, CL_FLOAT,
- CL_LUMINANCE_S8: CL_SIGNED_INT8, CL_UNSIGNED_INT8,
- CL_LUMINANCE1: CL_SIGNED_INT8, CL_UNSIGNED_INT8,
- CL_LUMINANCE1_2: CL_SIGNED_HALF_FLOAT, CL_UNSIGNED_HALF_FLOAT,
- CL_LUMINANCE1_4: CL_SIGNED_FLOAT, CL_UNSIGNED_FLOAT,
- CL_LUMINANCE8_8: CL_SIGNED_INT8, CL_UNSIGNED_INT8,
- CL_LUMINANCE8_10: CL_SIGNED_INT8, CL_UNSIGNED_INT8,
- CL_LUMINANCE8_12: CL_SIGNED_INT8, CL_UNSIGNED_INT8,
- CL_LUMINANCE16_16: CL_SIGNED_INT16, CL_UNSIGNED_INT16,
- CL_LUMINANCE16_32: CL_SIGNED_INT16, CL_UNSIGNED_INT32,
- CL_LUMINANCE32_32: CL_SIGNED_INT32, CL_UNSIGNED_INT32,
- CL_LUMINANCE6_16: CL_SIGNED_HALF_FLOAT, CL_UNSIGNED_INT16,
- CL_LUMINANCE6_32: CL_SIGNED_FLOAT, CL_UNSIGNED_INT32,
- CL_LUMINANCE10_10: CL_SIGNED_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE10_20: CL_SIGNED_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE12_12: CL_SIGNED_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE12_24: CL_SIGNED_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE12_32: CL_SIGNED_FLOAT, CL_UNSIGNED_INT32,
- CL_LUMINANCE16_24: CL_SIGNED_INT16, CL_HALF_FLOAT,
- CL_LUMINANCE16_48: CL_SIGNED_INT16, CL_HALF_FLOAT,
- CL_LUMINANCE16_64: CL_SIGNED_INT16, CL_HALF_FLOAT,
- CL_LUMINANCE16_128: CL_SIGNED_INT16, CL_HALF_FLOAT,
- CL_LUMINANCE16_256: CL_SIGNED_INT16, CL_HALF_FLOAT,
- CL_LUMINANCE32_64: CL_SIGNED_INT32, CL_HALF_FLOAT,
- CL_LUMINANCE32_128: CL_SIGNED_INT32, CL_HALF_FLOAT,
- CL_LUMINANCE32_256: CL_SIGNED_INT32, CL_HALF_FLOAT,
- CL_LUMINANCE64_64: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE64_128: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE64_256: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE128_128: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE256_256: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE512_512: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE1024_1024: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE2048_2048: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE4096_4096: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE8192_8192: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE16384_16384: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE32768_32768: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE65536_65536: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE131072_131072: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE262144_262144: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE524288_524288: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE1048576_1048576: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE2097152_2097152: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE4194304_4194304: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE8388608_8388608: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE16777216_16777216: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE33554432_33554432: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE67108864_67108864: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE134217728_134217728: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE268435456_268435456: CL_HALF_FLOAT, CL_HALF_FLOAT,
- CL_LUMINANCE536870912_536870912: CL_HALF_FLOAT, CL_HALF_FLOAT,
Image Read and Write Functions

The built-in functions defined in this section can only be used with image memory objects created
cUsing the OpenCL extension. sampler specifies the addressing and filtering mode to use. OpenCL refers to one of the access

- Writes to images with sRGB channel orders requires device support of the cl_khr_srgb_image_writes extension.
- read_imageh and write_imageh require the cl_khr_fp16 extension.
- MSAA images require the cl_khr_gl_msaa_sharing extension.
- Image 3D writes require the extension cl_khr_3d_image_writes.

Read and write functions for 2D images

Read an element from a 2D image, or write a color value to a location in a 2D image.

void write_image ((Qual image2d_t image, sampler_t sampler, int2, float4) coord)
void write_image ((Qual image2d_array_t image, int4 coord, float4) coord)
void write_image ((Qual image2d_array_depth_t image, int2 coord, half4 color)

Read and write functions for 3D images

Read an element from a 3D image, or write a color value to a location in a 3D image. Writing to 3D images requires the cl_khr_3d_image_writes extension.

void read_image ((Qual image3d_t image, sampler_t sampler, int4, float4) coord)
void read_image ((Qual image3d_t image, int4 coord, float4) coord)
void read_image ((Qual image3d_t image, int4 coord, uint4) coord)
void read_image ((Qual image3d_t image, sampler_t sampler, int4, float4) coord)
void read_image ((Qual image3d_t image, sampler_t sampler, int4, float4) coord)
void read_image ((Qual image3d_t image, sampler_t sampler, int4, float4) coord)
void read_image ((Qual image3d_t image, sampler_t sampler, int4, float4) coord)

Extended mipmap read and write functions

These functions require the cl_khr_mipmap_image and cl_khr_mipmap_image_writes extensions.

(Continued on next page >)
Query image Channel data type and order

- `get_image_channel_data_type` (aQual image1d_t image, int4)
- `get_image_channel_data_type` (aQual image2d_t image, int4)
- `get_image_channel_data_type` (aQual image3d_t image, int4)
- `get_image_channel_data_type` (aQual image2d_array_t image, int4)
- `get_image_channel_data_type` (aQual image3d_array_t image, int4)

Access Qualifiers [6.6]

Apply to 2D and 3D image types to declare if the image memory object is being read or written by a kernel.

- `__read_only`, `read_only`
- `__write_only`, `write_only`

Sampler Objects [5.7]

Items in blue require the cl_khr_mipmap_image extension.

- `cl_sampler clCreateSamplerWithProperties (cl_context context, const cl_sampler_properties *sampler_properties, cl_int *errcode_ret)`
- `cl_sampler_properties` (Table 5.15)
- `CL_SAMPLER_ADDRESSING_FILTER_MODE`, `CL_SAMPLER_MIP_FILTER_MODE`, `CL_SAMPLER_LOD_MIN_MAX`
- `cl_int cIEtainSampler (cl_sampler sampler)`
- `cl_int cIEleaseSampler (cl_sampler sampler)`
- `cl_int cIGetSamplerInfo (cl_sampler sampler, cl_sampler_info sampler_info, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`
- `param_name` (CL_SAMPLER_REFERENCE_COUNT, CL_SAMPLER_CONTEXT_FILTER_MODE, CL_SAMPLER_ADDRESSING_MODE, CL_SAMPLER_NORMALIZED_COORDS) (Table 5.16)


The sampler can be passed as an argument to the kernel using cIGetKernelArg, or can be declared in the outermost scope of kernel functions, or can be a constant variable of type sampler_t declared in the program source.

- `const sampler_t sampler_name = <normalized-mode> | <address-mode> | <filter-mode>`
- normalized-mode: `CL_NORMALIZED_COORDS_TRUE, FALSE`
- address-mode: `CL_ADDRESS_X`, where X may be NONE, REPEAT, CLAMP, CLAMP_TO_EDGE, MIRRORED_REPEAT
- filter-mode: `CL_FILTER_NEAREST, CL_FILTER_LINEAR`
OpenCL 2.1 Reference Card

Using OpenCL Extensions [9]
The following extensions extend the OpenCL API. Extensions shown in italics provide core features.
To control an extension: #pragma OPENCL EXTENSION extension_name : enable | disable
To test if an extension is supported, use clGetPlatformInfo() or clGetDeviceInfo()
To get the address of the extension function: clGetExtensionFunctionAddressForPlatform()

OpenCL Shading [9.5 - 9.7]
These functions require the cl_khr_gl_sharing or cl_khr_apple_gl_sharing extension.

CL Context > GL Context, Sharegroup [9.5.5]
- cl_int clGetGLContextInfoKHR
  - const cl_context_properties *properties,
  - cl_gl_context_info *param_name,
  - size_t *param_value_size, void *param_value,
  - size_t *param_value_size_ret
  - param_name: CL_DEVICES FOR GL CONTEXT_KHR,
  - CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR

CL Buffer Objects > GL Buffer Objects [9.6.2]
- cl_mem clCreateFromGLBuffer
  - cl_context context, glBuffer *buffer, cl_int *errcode_ret
  flags: GL_MEM_READ_ONLY, GL_MEM_WRITE_ONLY, GL_MEM_READ_WRITE

CL Image Objects > GL Textures [9.6.3]
- cl_mem clCreateFromGLTexture
  - cl_context context, cl_mem *memobj, glTextureImage *image, cl_int *errcode_ret
  flags: GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D

DX9 Media Surface Sharing [9.9]
The header file is cl_d9_media_sharing.h. Enable the extension cl_khr_d9_media-sharing.
- cl_int clGetDeviceIDsFromDX9MediaAdapterKHR(cl_platform_id platform, cl_uint num_media_adapters, cl_d9_media_adapter_typeKHR *media_adapters, cl_int *errcode_ret)
  flags: cl_khrisdiction_memory

EGL Interoperability [9.18, 9.19]
Create CL Image Objects from EGL
These functions require the extension cl_khr_eglu_image.
- cl_mem clCreateFromEGLImageKHR(cl_context context, EGLDisplay display, EGLImageKHR image, cl_mem *memflags, clsetImage_propertiesKHR *properties, cl_int *errcode_ret)

Direct3D 10 Sharing [9.8.7]
These functions require the cl_khr_d3d10_sharing extension. The associated header file is cl_d3d10.h.
- cl_int clGetDeviceIDsFromD3D10KHR(cl_platform_id platform, cl_d3d10_device_sourceKHR *device_source, cl_uint *errcode_ret)
  flags: cl_khr_device_enque_local_arg_types

Create CL Event Objects from EGL
This function requires the extension cl_khr_event
- cl_event clCreateFromEGLErrorKHR(cl_context context, CL.showErrorKHR sync, EGLDisplayKHR display, cl_int *errcode_ret)
Example of Enqueuing Kernels

Arguments that are a pointer type to local address space [6.13.17.2]
A block passed to enqueue_kernel can have arguments declared to be a pointer to local memory. The enqueue_kernel built-in function variants allow blocks to be enqueued with a variable number of arguments. Each argument must be declared to be a void pointer to local memory. These enqueue_kernel built-in function variants also have a corresponding number of arguments each of type uint that follow the block argument. These arguments specify the size of each local memory pointer argument of the enqueued block.

```
kernel void
my_func_A_local_arg1(global int *a, local int *lptr, ...)
{
  ...
}

kernel void
my_func_A_local_arg2(global int *a,
  local int *lptr1, local float4 *lptr2, ...)
{
  ...
}

kernel void
my_func_B(global int *a, ...)
{
  ...
  ndrange_t ndrange = ndrange_1d(...);
  uint local_mem_size = compute_local_mem_size();
  enqueue_kernel(get_default_queue(),
    CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
    ndrange,
    ^(local void *)(
       my_func_A_local_arg1(a, (local int *)p, ...),
       local_mem_size);
}

kernel void
my_func_C(global int *a, ...)
{
  ...
  ndrange_t ndrange = ndrange_1d(...);
  uint local_mem_size = compute_local_mem_size();
  enqueue_kernel(get_default_queue(),
    CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
    ndrange,
    ^(local void *)(
       my_func_A_local_arg2(
         a,
         (local int *)lptr1,
         (local float4 *)lptr2, ...));
  // calculate local memory size for lptr
  // argument in local address space for myblk_A
  uint local_mem_size = compute_local_mem_size();
  enqueue_kernel(get_default_queue(),
    CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
    ndrange,
    myblk_A, local_mem_size, local_mem_size * 4);
```

A Complete Example [6.13.17.3]
The example below shows how to implement an iterative algorithm where the host enqueues the first instance of the nd-range kernel (dp_func_A). The kernel dp_func_A will launch a kernel (evaluate_dp_work_A) that will determine if new nd-range work needs to be performed. If new nd-range work does need to be performed, then evaluate_dp_work_A will enqueue a new instance of dp_func_A. This process is repeated until all the work is completed.

```
kerneld void
dp_func_A(queue_t q, ...)
{
  ...
  // queue a single instance of evaluate_dp_work_A to
  // device queue q. queued kernel begins execution after
  // kernel dp_func_A finishes
  if (get_global_id(0) == 0)
  {
    enqueue_kernel(q,
      CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
      ndrange_1d(1),
      ^(evaluate_dp_work_A(q, ...));
  }
}

kernel void
evaluate_dp_work_A(queue_t q, ...)
{
  // check if more work needs to be performed
  bool more_work = check_new_work(...);
  if (more_work)
  {
    size_t global_work_size = compute_global_size(...);
    void (^dp_func_A_blk)(void) =
      ^(dp_func_A(q, ...));
    // get local WG-size for kernel dp_func_A
    size_t local_work_size =
      get_kernel_work_group_size(dp_func_A_blk);
    // build nd-range descriptor
    ndrange_t ndrange = ndrange_1D(global_work_size,
      local_work_size);
    // enqueue dp_func_A
    enqueue_kernel(q,
      CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
      ndrange,
      dp_func_A_blk);
  }
}
```