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 computer servers, desktop computer systems, and handheld devices.

Specifications and online references are available at www.khronos.org/opencl.

OpenCL API Reference

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL device functionality, 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

- **cl_int cGetPlatformIDs** (cl_uint num_entries, cl_platform_id *platforms, cl_uint *num_platforms)
- **cl_int cGetPlatformInfo** (cl_platform_id platform, cl_platform_info name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)
- **cl_int cGetDeviceInfo** (cl_device_id device, cl_device_info info_name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)

Partitioning a Device

- **cl_int cCreateCommandQueueWithProperties** (cl_context context, cl_device_id device, const cl_command_queue_properties *properties, cl_int *errcode_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

- **cl_command_queue cCreateCommandQueueWithProperties** (cl_context context, cl_device_id device, const cl_command_queue_properties *properties, cl_int *errcode_ret)

Contexts

- **cl_int cCreateContext** (const cl_context_properties *properties, cl_uint num_devices, const cl_device_id *devices, void *(CL_CALLBACK fn_notify) (const char *errinfo, const void *private_info, size_t cb, void *user_data), void *user_data, cl_int *errcode_ret)

Get CL Extension Function Pointers

- void *(clGetExtensionFunctionAddressForPlatform)** (cl_platform_id platform, const char *funcname)
Buffer Objects
Elements of buffer objects are stored sequentially and accessed using a pointer by a kernel executing on a device.

Create Buffer Objects [5.2.1]
c_mem clCreateBuffer (cl_context context, cl_mem_flags flags, size_t size, void *host_ptr, cl_int errcode_ret)
flags: (Table 5.5): CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY, CL_MEM_HOST_NO_ACCESS, CL_MEM_HOST_ACCESS, CL_MEM_USE_ALLOC, COPY_HOST_PTR
returns: cl_mem

Read, Write, Copy, Fill Buffer Objects [5.2.2-3]
c_int clEnqueueReadBuffer...(command, flags, size_t offset, size_t size, void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
c_int clEnqueueWriteBuffer...(command, flags, size_t offset, size_t size, void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
c_int clEnqueueWriteBufferRect...(command, flags, size_t offset, size_t size, cl_mem buffer, void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
c_int clEnqueueFillBuffer...(command, flags, size_t offset, size_t size, void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
c_int clEnqueueCopyBuffer...(command, flags, size_t offset, size_t size, cl_mem src_buffer, cl_mem dst_buffer, void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
c_int clEnqueueCopyBufferRect...(command, flags, size_t offset, size_t size, cl_mem src_buffer, cl_mem dst_buffer, void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

Map Buffer Objects [5.2.4]
void *clEnqueueMapBuffer...
flags: CL_MAP_READ, CL_MAP_WRITE_INVALIDATE_REGION

Memory Objects
A memory object is a handle to a reference counted region of global memory. Includes Buffer Objects, Image Objects, and Pipe Objects. Items in blue apply when the appropriate extension is supported.

Memory Objects [5.5.1, 5.5.2]
c_int clRetainMemObject (cl_mem memobj)
c_int clReleaseMemObject (cl_mem memobj)
c_int clSetMemObjectDestructorCallback (cl_mem memobj, void *(fn_ptr) notify) (fn_ptr notify)
c_int clEnqueueUnmapMemObject (cl_command_queue command_queue, cl_mem memobj, void *mapped_ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

Migrate Memory Objects [5.5.4]
c_int clEnqueueMigrateMemObjects (cl_command_queue command_queue, cl_uint num_mem_objects, cl_mem_migration_flags flags, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

OpenCL Device Architecture Diagram
The table below shows memory regions with allocation and memory access capabilities.

<table>
<thead>
<tr>
<th>Region</th>
<th>Host Allocation</th>
<th>Host Access</th>
<th>Kernel Allocation</th>
<th>Kernel Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>No allocation</td>
<td>R/W access</td>
<td>Static allocation</td>
<td>R-only access</td>
</tr>
<tr>
<td>Constant</td>
<td>Static allocation</td>
<td>R/W access</td>
<td>Static allocation</td>
<td>R/W access</td>
</tr>
<tr>
<td>Local</td>
<td>No allocation</td>
<td>Static allocation</td>
<td>R/W access</td>
<td></td>
</tr>
<tr>
<td>Private</td>
<td>No allocation</td>
<td>Static allocation</td>
<td>R/W access</td>
<td></td>
</tr>
</tbody>
</table>

OpenCL Class Diagram
The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language (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
- Relationships
  - abstract classes
  - aggregations
  - inheritance
  - relationship navigability

Cardinality
- many *
- one and only one
- optionally one 0..1
- one or more 0..*

Conversions and Type Casting Examples [6.2]
- T a = T(b); // Scalar to scalar
- R a = convert_T(b); // or scalar to vector
- R one of the following rounding modes:
  - _rtt to nearest even
  - _rzu toward zero
  - _rtp toward + infinity
  - _rtn toward - infinity

Pipe Object Queries [5.4.2]
c_int clGetPipeInfo (cl_mem pipe, cl_pipe_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)
param_name: CL_PIPE_PACKET_SIZE
flags: CL_MIGRATE_MEM_OBJECT_HOST, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

Query Memory Object [5.5.5]
c_int clGetMemObjectInfo (cl_mem obj, cl_mem_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)
param_name: CL_MEM_OFFSET, CL_MEM_SIZE, CL_MEM_FLAGS, CL_MEM_TYPE, CL_MEM_HOST_PTR, CL_MEM_HOST_NO_ACCESS, CL_MEM_FLAGS
flags: CL_MEM_ASSOCIATED_MEMOBJECT, CL_MEM_DEV_ACCESSIBLE, CL_MEM_MAP_READ, CL_MEM_MAP_WRITE, CL_MEM_MAP_WRITE_INVALIDATE_REGION, CL_MEM_WRITE_INVALIDATE_REGION, CL_MEM_READ_INVALIDATE_REGION

Pipe
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 [5.4.4]
c_mem clCreatePipe (cl_context context, cl_mem_flags flags, cl_uint packet_size, cl_uint max_packets, const cl_pipe_properties *properties, cl_int errcode_ret)
flags: 0 or CL_MEM_READ_WRITE, CL_MEM_READ_WRITE_ONLY, CL_MEM_HOST_NO_ACCESS

Pipe Object Queries [5.4.2]
c_int clGetPipeInfo (cl_mem pipe, cl_pipe_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)
param_name: CL_PIPE_PACKET_SIZE, CL_PIPE_MAX_PACKETS
flags: CL_MIGRATE_MEM_OBJECT_HOST, CL_MIGRATE_MEM_OBJECT_CONTENT_UNDEFINED

Query Memory Object [5.5.5]
c_int clGetMemObjectInfo (cl_mem obj, cl_mem_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)
OpenCL 2.1 Reference Guide

Shared Virtual Memory

Shared Virtual Memory (SVM) allows the host and kernels executing on devices to directly share complex, pointer-containing data structures such as trees and linked lists. See more on SVM on page 4 of this reference guide.

SVM Sharing Granularity [5.6.1]*

void *clSVMAlloc (cl_context context, size_t size, cl_uint alignment)

flags:
- CL_MEM_READ_WRITE,
- CL_MEM_WRITE_ONLY,
- CL_MEM_FINE_GRAIN_BUFFER,
- CL_MEM_ATOMICS

void clSVMFree (cl_context context, void *svm_pointer)

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 [5.8.1]

cl_program clCreateProgramWithSource (cl_context context, const char *source, size_t length, cl_int *errcode_ret)

cl_program clCreateProgramWithIL (cl_context context, const char *program, size_t length, cl_int *errcode_ret)

cl_program clCreateProgramWithBinary (cl_context context, cl_uint num_devices, cl_device_id *device_list, const size_t *lengths, const unsigned char **binaries, cl_int *binary_status, cl_int *errcode_ret)

cl_program clCreateProgramWithBuiltInKernels (cl_context context, cl_uint num_devices, cl_device_id *device_list, cl_int *kernel_names, cl_int *errcode_ret)

cl_int clReleaseProgram (cl_program program)

Unload the OpenCL Compiler [5.8.6]

cl_int clUnloadPlatformCompiler (cl_platform_id platform)

Query Program Objects [5.8.7]

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_list, cl_int *param_value_size, void *param_value, size_t *param_value_size_ret)

Optimization options:
- -cl-opt-disable
- -cl-unsafe-math-optimizations
- -cl-fast-relaxed-math

Warning request/suppression:
- -w
- -Werr

Control OpenCL C language version:
- -cl-version=OPENCL X.X

Library linking options:
- -create-library
- -enable-link-options

Flush and Finish [5.15]

cl_int clFlush (cl_command_queue command_queue)

cl_int clFinish (cl_command_queue command_queue)

Shared Virtual Memory

Shared Virtual Memory (SVM) allows the host and kernels executing on devices to directly share complex, pointer-containing data structures such as trees and linked lists. See more on SVM on page 4 of this reference guide.

SVM Sharing Granularity [5.6.1]

void *clSVMAlloc (cl_context context, size_t size, cl_uint alignment)

flags:
- CL_MEM_READ_WRITE,
- CL_MEM_WRITE_ONLY,
- CL_MEM_FINE_GRAIN_BUFFER,
- CL_MEM_ATOMICS

void clSVMFree (cl_context context, void *svm_pointer)

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 [5.8.1]

cl_program clCreateProgramWithSource (cl_context context, const char *source, size_t length, cl_int *errcode_ret)

cl_program clCreateProgramWithIL (cl_context context, const char *program, size_t length, cl_int *errcode_ret)

cl_program clCreateProgramWithBinary (cl_context context, cl_uint num_devices, cl_device_id *device_list, const size_t *lengths, const unsigned char **binaries, cl_int *binary_status, cl_int *errcode_ret)

cl_program clCreateProgramWithBuiltInKernels (cl_context context, cl_uint num_devices, cl_device_id *device_list, cl_int *kernel_names, cl_int *errcode_ret)

cl_int clReleaseProgram (cl_program program)

Unload the OpenCL Compiler [5.8.6]

cl_int clUnloadPlatformCompiler (cl_platform_id platform)

Query Program Objects [5.8.7]

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_list, cl_int *param_value_size, void *param_value, size_t *param_value_size_ret)

Optimization options:
- -cl-opt-disable
- -cl-unsafe-math-optimizations
- -cl-fast-relaxed-math

Warning request/suppression:
- -w
- -Werr

Control OpenCL C language version:
- -cl-version=OPENCL X.X

Library linking options:
- -create-library
- -enable-link-options

Flush and Finish [5.15]

cl_int clFlush (cl_command_queue command_queue)

cl_int clFinish (cl_command_queue command_queue)
Kernel Objects (continued)

cl_int c1GetKernelSubGroupInfo (cl_int kernel, cl_device_id device, cl_kernel_sub_group_info param_name, size_t input_value_size, const void *input_value, size_t t param_value_size, void *param_value, size_t t param_value_size_ret)

param_name: [Table 5.22]
CL_KERNEL_LOCAL_SIZE_FOR_SUB_GROUP_COUNT, CL_KERNEL_MAX_SUB_GROUP_SIZE_FOR_NDARGS, CL_KERNEL_SUB_GROUP_COUNT_FOR_NDARGS

Event Objects

Event objects can be used to refer to a kernel execution command, and read, write, map, and copy commands on memory objects or user events.

Event Objects [5.11]
cl_event c1CreateUserEvent(cl_context context, cl_int *errcode_ret)
cl_event c1SetUserEventStatus(cl_event event, cl_int execution_status)
cl_event c1WaitForEvents(cl_uint num_events, const cl_event *event_list)

Execute Kernels [5.10]
cl_int c1EnqueueNDRangeKernel (cl_command_queue command_queue, cl_kernel kernel, cl_uint work_dim, const size_t t *global_work_offset, const size_t t *global_work_size, const size_t t *local_work_size, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)

cl_int c1EnqueueNativeKernel (cl_command_queue command_queue, void (CL_CALLBACK *user_func)(void *), void *args, size_t cb args, cl_uint num_mem_objects, const cl_mem *mem_list, const void **args_mem_loc, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)

Markers, Barriers, Waiting for Events [5.12]
cl_int c1EnqueueMarkerWithWaitList (cl_command_queue command_queue, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)
cl_int c1EnqueueBarrierWithWaitList (cl_command_queue command_queue, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)

Proiling Operations [5.14]
cl_int c1GetEventProfilingInfo (cl_event event, cl_profiling_info param_name, size_t t param_value_size, void *param_value, size_t t param_value_size_ret)

param_name: [Table 5.25]
CL_PROFILING_COMMAND_{SUBMIT, START, END}, CL_PROFILING_COMMAND_QUEUED, CL_PROFILING_COMMAND_COMPLETE

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.

Summary of SVM Options in OpenCL [3.3.3, Table 3-2]

<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>

Notes
## OpenCL C Language Reference

### Section and table references are to the OpenCL C Language 2.0 specification.

#### OpenCL 2.1 Reference Guide

**Supported Data Types**
The optional 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>OPTIONAL</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>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>ptrdiff_t</td>
<td>--</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>intptr_t</td>
<td>--</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>uintptr_t</td>
<td>--</td>
<td>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>void</td>
<td>void</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>charn</td>
<td>cl_charn</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>ucharn</td>
<td>cl_uchar</td>
<td>8-bit unsigned</td>
</tr>
<tr>
<td>shortn</td>
<td>cl_short</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>ushortn</td>
<td>cl_ushort</td>
<td>16-bit unsigned</td>
</tr>
<tr>
<td>intn</td>
<td>cl_int</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>uintn</td>
<td>cl_uint</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>longn</td>
<td>cl_long</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>ulongn</td>
<td>cl_ulong</td>
<td>64-bit unsigned</td>
</tr>
<tr>
<td>floatn</td>
<td>cl_float</td>
<td>32-bit float</td>
</tr>
<tr>
<td>doublen</td>
<td>OPTIONAL</td>
<td>64-bit float</td>
</tr>
<tr>
<td>halfn</td>
<td>cl_half</td>
<td>16-bit float (storage only)</td>
</tr>
<tr>
<td>size_t</td>
<td>--</td>
<td>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>ptrdiff_t</td>
<td>--</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>intptr_t</td>
<td>--</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>uintptr_t</td>
<td>--</td>
<td>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>void</td>
<td>void</td>
<td>void</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>bools</td>
<td>boolean vector</td>
</tr>
<tr>
<td>halfn</td>
<td>16-bit, vector</td>
</tr>
<tr>
<td>quadn, quadn</td>
<td>128-bit float, vector</td>
</tr>
<tr>
<td>complexn, halfn</td>
<td>complex float, vector</td>
</tr>
<tr>
<td>imaginaryn, imaginaryhalf</td>
<td>imaginary float, imaginary half</td>
</tr>
<tr>
<td>complexfloatn, complexfloatn</td>
<td>complex float, complex float</td>
</tr>
<tr>
<td>imaginairyfloatn, imaginairyfloatn</td>
<td>imaginary float, imaginary float</td>
</tr>
<tr>
<td>complexquadn, complexquadn</td>
<td>complex double, complex double</td>
</tr>
<tr>
<td>imaginairyquadn, imaginairyquadn</td>
<td>imaginary double, imaginary double</td>
</tr>
<tr>
<td>floatnmx, floatnmx</td>
<td>n*m matrix of 32-bit floats</td>
</tr>
<tr>
<td>doublingmx</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>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>v.x, v.s0</td>
<td>v.y, v.s1</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>v.x, v.s0</td>
<td>v.y, v.s1</td>
<td>v.z, v.s2</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>v.x, v.s0</td>
<td>v.y, v.s1</td>
<td>v.z, v.s2</td>
<td>v.w, v.s3</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>v.s0</td>
<td>v.s1</td>
<td>v.s2</td>
<td>v.s3</td>
<td>v.s4</td>
<td>v.s5</td>
<td>v.s6</td>
<td>v.s7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float16 v;</td>
<td>v.s0</td>
<td>v.s1</td>
<td>v.s2</td>
<td>v.s3</td>
<td>v.s4</td>
<td>v.s5</td>
<td>v.s6</td>
<td>v.s7</td>
<td>v.s8</td>
<td>v.s9</td>
<td>v.sa</td>
<td>v.sb</td>
<td>v.sc</td>
<td>v.sd</td>
<td>v.se</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.: v.xy, v.xx, v.xo.

<table>
<thead>
<tr>
<th>v.lo</th>
<th>v.hi</th>
<th>v.odd</th>
<th>v.even</th>
</tr>
</thead>
<tbody>
<tr>
<td>v.x</td>
<td>v.xs</td>
<td>v.y</td>
<td>v.yz</td>
</tr>
<tr>
<td>v.z</td>
<td>v.zs</td>
<td>v.w</td>
<td>v.wx</td>
</tr>
<tr>
<td>v.s0</td>
<td>v.s1</td>
<td>v.s2</td>
<td>v.s3</td>
</tr>
<tr>
<td>v.s4</td>
<td>v.s5</td>
<td>v.s6</td>
<td>v.s7</td>
</tr>
<tr>
<td>v.s8</td>
<td>v.s9</td>
<td>v.sa</td>
<td>v.sb</td>
</tr>
<tr>
<td>v.sc</td>
<td>v.sd</td>
<td>v.se</td>
<td>v.sf</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:

| + | - | * | % | / | -- |
| ++ | == | != | && | ^ | ^ |
| > | >= | <= | || | >= | <= |

**Address Space Qualifiers [6.5]**

- global, global
- local, local
- constant, constant
- private, private

**Function Qualifiers [6.7]**

- _kernel_, _kernel_
- _attribute_ (vec_type_hint(type))
- _attribute_ (vec_type(type))
- _attribute_ (vec_type_hint(type))
- _attribute_ (vec_type(type))
- _attribute_ (vec_type(type))
- _attribute_ (vec_type(type))
- _attribute_ (vec_type(type))
- _attribute_ (vec_type(type))

**Attribute Qualifiers [6.11]**

Use to specify special attributes of enum, struct, and union types.

- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)

Use to specify special attributes of variables or structure fields.

- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)
- _attribute_ (alignedalignment)

Use to specify basic blocks and control-flow-statements.

- _attribute_ (att1)(...)
- _attribute_ (att1)(...)
- _attribute_ (att1)(...)
- _attribute_ (att1)(...)
- _attribute_ (att1)(...)
- _attribute_ (att1)(...)
- _attribute_ (att1)(...)
- _attribute_ (att1)(...)
- _attribute_ (att1)(...)

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

- _attribute_ (opencl_unroll(n))
- _attribute_ (opencl_unroll(n))
- _attribute_ (opencl_unroll(n))
- _attribute_ (opencl_unroll(n))
- _attribute_ (opencl_unroll(n))
- _attribute_ (opencl_unroll(n))
- _attribute_ (opencl_unroll(n))
- _attribute_ (opencl_unroll(n))
- _attribute_ (opencl_unroll(n))

**Preprocessor Directives & Macros [6.10]**

```c
#pragma OPENCL_FP_CONTRACT on-off-switch
#on-off-switch; ON, OFF, DEFAULT

_FILE_
_FUNC_
_LINE_
_OPENCL_VERSION_
_CL_VERSION_1_0
_CL_VERSION_1_1
_CL_VERSION_1_2
_CL_VERSION_2_0
_OPENCL_C_VERSION_
_ENDIAN_LITTLE_
_IMAGE_SUPPORT_
_FAST_RELAXED_MATH_
_FP_FAST_FMA
_FP_FAST_FMAF
_FP_FAST_FMA_HALF
_kernel_exec (X, typen)
_attribute_ (work_group_size(X, Y, Z))
_attribute_ (vec_type_hint(type))
```

©2015 Khronos Group - Rev. 1118

www.khronos.org/opengl
### 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 `int`.
4. The argument is named "num."
5. Multiplier captured from block's environment.

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

### Work-Item Built-in Functions [6.13.1] [9.4.2]
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
size_t get_work_dim () Number of dimensions in use
size_t get_global_index ( uint dimind ) Number of global work-items
size_t get_global_id ( uint dimind ) Global work-item ID value
size_t get_local_index ( uint dimind ) Number of local work-items if kernel executed with uniform work-group size
size_t get_local_id ( uint dimind ) Local work-item ID
size_t get_num_groups ( uint dimind ) Number of work-groups

size_t get_group_id ( uint dimind ) Work-group ID
size_t get_global_offset ( uint dimind ) Global offset
size_t get_global_linear_id ( uint dimind ) Work-items 1-dimensional global ID
size_t get_local_linear_id ( uint dimind ) Work-items 1-dimensional local ID
int get_sub_group_id () Number of work-items in the sub-group
size_t get_max_sub_group_id () Number of subgroups
size_t get_num_sub_groups () Number of subgroups
size_t get_sub_group_id ( uint subgroup_id ) Sub-group ID
size_t get_sub_group_local_id () Unique work-item ID
```

### Math Built-in Functions [6.13.2] [9.4.2]
For F is type float, optionally double, or half if the cl_khr_fp16 extension is enabled. Tn is the vector form of Tn where n is 2, 3, 4, 8, or 16. T is T 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 the float or float variants.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acos(T)</code></td>
<td>Arc cosine</td>
</tr>
<tr>
<td><code>acos(fh)</code></td>
<td>Arc hyperbolic cosine</td>
</tr>
<tr>
<td><code>acosl(T)</code></td>
<td>Arc cosine (n)</td>
</tr>
<tr>
<td><code>asin(T)</code></td>
<td>Arc sine</td>
</tr>
<tr>
<td><code>asinh(T)</code></td>
<td>Arc hyperbolic sine</td>
</tr>
<tr>
<td><code>asins(T)</code></td>
<td>Arc hyperbolic sine (n)</td>
</tr>
<tr>
<td><code>atan(T)</code></td>
<td>Arc tangent</td>
</tr>
<tr>
<td><code>atan2(T)</code></td>
<td>Arc tangent of y/x</td>
</tr>
<tr>
<td><code>atanh(T)</code></td>
<td>Hyperbolic arc tangent</td>
</tr>
<tr>
<td><code>atanhl(T)</code></td>
<td>Hyperbolic arc tangent (n)</td>
</tr>
<tr>
<td><code>cbrtl(T)</code></td>
<td>Cube root</td>
</tr>
<tr>
<td><code>ceil(T)</code></td>
<td>Round to integer toward + infinity</td>
</tr>
<tr>
<td><code>copy_sign(T, Y)</code></td>
<td>x with sign changed to sign of y</td>
</tr>
<tr>
<td><code>cos(T)</code></td>
<td>Cosine</td>
</tr>
<tr>
<td><code>cosh(T)</code></td>
<td>Hyperbolic cosine</td>
</tr>
<tr>
<td><code>cospl(T)</code></td>
<td>Hyperbolic cosine (n)</td>
</tr>
<tr>
<td><code>half_divide(T, T)</code></td>
<td>x / y</td>
</tr>
<tr>
<td><code>native_divide(T, T)</code></td>
<td>x / y (T may only be float or float)</td>
</tr>
<tr>
<td><code>erf(T)</code></td>
<td>Complementary error function</td>
</tr>
<tr>
<td><code>erf(T)</code></td>
<td>Error function of T</td>
</tr>
<tr>
<td><code>exp(T)</code></td>
<td>Exponential base e</td>
</tr>
<tr>
<td><code>exp2(T)</code></td>
<td>Exponential base 2</td>
</tr>
<tr>
<td><code>expm1(T)</code></td>
<td>Exponential base 10</td>
</tr>
<tr>
<td><code>fabs(T)</code></td>
<td>Absolute value</td>
</tr>
<tr>
<td><code>fdim(T, T)</code></td>
<td>Positive difference between x and y</td>
</tr>
<tr>
<td><code>floor(T)</code></td>
<td>Round to integer towards infinity</td>
</tr>
<tr>
<td><code>fma(T, a, T, b)</code></td>
<td>Multiply and add, then round</td>
</tr>
<tr>
<td><code>fmax(T, T)</code></td>
<td>Return y if x &lt; y, otherwise it returns x</td>
</tr>
<tr>
<td><code>fmin(T, T)</code></td>
<td>Return y if x &lt; y, otherwise it returns x</td>
</tr>
<tr>
<td><code>fnextafter(T, T)</code></td>
<td>Next representable floating-point value after x in the direction of y</td>
</tr>
<tr>
<td><code>fpow(T, T)</code></td>
<td>Compute x to the power of y</td>
</tr>
<tr>
<td><code>fround(T)</code></td>
<td>Integer nearest to y</td>
</tr>
<tr>
<td><code>fround(T)</code></td>
<td>Integer nearest to y</td>
</tr>
<tr>
<td><code>half_recip(T)</code></td>
<td>1 / x</td>
</tr>
<tr>
<td><code>native_recip(T)</code></td>
<td>(T may only be float or float)</td>
</tr>
<tr>
<td><code>remainder(T)</code></td>
<td>Floating point remainder</td>
</tr>
<tr>
<td><code>remqu(T)</code></td>
<td>Remainder and quotient</td>
</tr>
<tr>
<td><code>rint(T)</code></td>
<td>Round to nearest even integer</td>
</tr>
<tr>
<td><code>rootn(T, n)</code></td>
<td>Compute x to the power of 1/y</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXFLOAT</td>
<td>Value of maximum non-infinite single-precision floating-point number</td>
</tr>
<tr>
<td>HUGE_VAL</td>
<td>Positive float expression, evaluates to +infinity</td>
</tr>
<tr>
<td>HUGE_VALF</td>
<td>Positive double expression, evaluates to +infinity</td>
</tr>
<tr>
<td>INFINITY</td>
<td>Constant float expression, positive or unsigned infinity</td>
</tr>
<tr>
<td>NAN</td>
<td>Constant float expression, quiet NaN</td>
</tr>
</tbody>
</table>

When double precision is supported, macros ending in _F are available in type double by replacing _F with _H.
Integer Built-in Functions [6.13.3]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T abs(T x)</code></td>
<td>Absolute value of <code>x</code></td>
</tr>
<tr>
<td><code>T abs_diff(T x, T y)</code></td>
<td>Difference between <code>x</code> and <code>y</code> without modulo overflow</td>
</tr>
<tr>
<td><code>T add_sat(T x, T y)</code></td>
<td>Saturates the result if <code>x</code> and <code>y</code> add to a value outside the range of <code>T</code></td>
</tr>
<tr>
<td><code>T add&lt;T a, T b, T c&gt;</code></td>
<td>Saturates the result if <code>a</code> and <code>b</code> add to a value outside the range of <code>T</code></td>
</tr>
<tr>
<td><code>T clamp(T x, T y, T z)</code></td>
<td>Clamps <code>x</code> to the range defined by <code>y</code> and <code>z</code></td>
</tr>
<tr>
<td><code>T clamp_u(T x, T max)</code></td>
<td>Clamps <code>x</code> to non-negative integers</td>
</tr>
<tr>
<td><code>T clamp&lt;T a, T b, T c&gt;</code></td>
<td>Clamps <code>x</code> to the range defined by <code>a</code>, <code>b</code>, and <code>c</code></td>
</tr>
<tr>
<td><code>T cross_product(T x, T y)</code></td>
<td>Cross product of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>T dot_product(T x, T y)</code></td>
<td>Dot product of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>T distances(T x)</code></td>
<td>Vector distance between <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>T dot(T x)</code></td>
<td>Dot product of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>T length(T x)</code></td>
<td>Vector length of <code>x</code></td>
</tr>
<tr>
<td><code>T normalize(T x)</code></td>
<td>Normalizes <code>x</code></td>
</tr>
</tbody>
</table>

For upsample, return type is scalar when the parameters are scalar.

The following fast integer functions optimize the performance of kernels. In these functions, `T` is type `int`, `uint`, `intr`, or `intrn`, where `n` is 2, 3, 4, 8, or 16. `half` and `halfn` types require the cl_khr_fp64 extension [9.4.5].

Relational Built-in Functions [6.13.6]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T isequal(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> and <code>y</code> are equal, else 0</td>
</tr>
<tr>
<td><code>T isneq(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> and <code>y</code> are not equal, else 0</td>
</tr>
<tr>
<td><code>T isgt(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> is greater than <code>y</code>, else 0</td>
</tr>
<tr>
<td><code>T islt(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> is less than <code>y</code>, else 0</td>
</tr>
<tr>
<td><code>T isge(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> is greater than or equal to <code>y</code>, else 0</td>
</tr>
<tr>
<td><code>T iseql(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> and <code>y</code> are equal, else 0</td>
</tr>
<tr>
<td><code>T isgneq(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> and <code>y</code> are not equal, else 0</td>
</tr>
<tr>
<td><code>T isgt_eq(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> is greater than or equal to <code>y</code>, else 0</td>
</tr>
<tr>
<td><code>T islt_eq(float x, float y)</code></td>
<td>Returns 1 if <code>x</code> is less than or equal to <code>y</code>, else 0</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 cl_khr_fp64 is enabled. `Tn` is the vector form of `T`, where `n` is 2, 3, 4, 8, or 16. `T` is `Tn` and `Tn`.

Geometric Built-in Functions [6.13.5] [9.4.4]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T distance(T x)</code></td>
<td>Distance between <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>T dot(T x)</code></td>
<td>Dot product of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>T length(T x)</code></td>
<td>Vector length of <code>x</code></td>
</tr>
<tr>
<td><code>T normalize(T x)</code></td>
<td>Normalizes <code>x</code></td>
</tr>
</tbody>
</table>

Cross product

- `float (3,4) cross (float3,4) p0, float3,4 p1, float (3,4) n1` | Vector product
- `half (3,4) cross (half3,4) p0, half3,4 p1, half (3,4) n1` | Vector product

Vector distance

- `float fast_distance (float0, float p0, float p1)` | Vector distance
- `float fast_length (float p)` | Vector length
Vector Data Load/Store [6.13.7] [9.4.6]

T is type char, uchar, short, ushort, int, uint, long, ulong, or float, optionally double, or half if the cl_khr_fp16 extension is enabled. Tn refers to the vector form of type T, where n is 2, 3, 4, 8, or 16. R defaults to current rounding mode, or is one of the rounding modes listed in 6.2.3.2.

| In vloadn (size_t t_offset, const [constant] * p) | Read vector data from address (p + (offset * n)) |
| void vstoren (Tn (size_t t_offset, size_t t_offset, const [constant] half * p) | Write a half to address (p + (offset * n)) |
| float vload_half (size_t t_offset, const [constant] half * p) | Read a half from address (p + (offset * n)) |
| float vload_32half (size_t t_offset, const [constant] half * p) | Read a half from address (p + (offset * n)) |

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_ T 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 ptrdiff_t. The type atomic_ T is a 32-bit integer. Atomic_ long and atomic_ ulong require extension cl_khr_int64_base_atomics or cl_khr_int64_extended_atomics. The atomic_double type requires double precision support. The default scope is work_group for local atomic and all _vmem devices for global atomics. The extensions cl_khr_int64_base_atomics and cl_khr_int64_extended_atomics 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_order, memory_scope, and memory_flag.

| void atomic_init (volatile A *obj, C value) | Initializes the atomic object pointed to by obj to the value value. |
| void atomic_store (volatile A *object, C desired) | Atomically replace the value pointed to by object with the value desired. Memory is affected according to the value of order. |
| void atomic_store_explicit (volatile A *object, C desired, memory_order order, memory_scope scope) | Atomically replace the value pointed to by object with the value desired. Memory is affected according to the value of order. |
| void atomic_exchange (volatile A *object, C desired) | Atomically replaces the value pointed to by object with the value desired. Memory is affected according to the value of order. |
| void atomic_exchange_explicit (volatile A *object, C desired, memory_order order, memory_scope scope) | Atomically replaces the value pointed to by object with the value desired. Memory is affected according to the value of order. |
| void atomic_compare_exchange (volatile A *object, C expected, C desired) | 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. |
| void atomic_fetch_add (volatile A *object, M operand) | 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. |

Synchronization & Memory Fence Functions [6.13.8]

flags argument is the memory address space, set to 0 or an OR’d combination of CL_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 flags, memory_scope scope) | Work-items in a work-group must execute this before any can continue |
| void atomic_work_item_fence (cl_mem_fence_flags flags, memory_scope scope) | Orders loads and stores of a work-item executing a kernel |
| void sub_group_barrier (cl_mem_fence_flags flags, memory_scope scope) | Work-items in a sub-group must execute this before any can continue |

Async Copies and Prefetch [6.13.10] [9.4.7]

T is type char, charr, uchar, short, short, ushort, ushort, int, intr, uint, uint, long, long, ulong, ulong, float, float, optionally double or double, or half or half if the cl_khr_fp16 extension is enabled.

| void atomic_flag_test_and_set (volatile atomic_flag *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. |
| void atomic_flag_clear (volatile atomic_flag *object) | Atomically sets the value pointed to by object to false. The order argument shall not be memory_order_acq_rel or memory_order_release. Memory is affected according to the value of order. |

Values for key for atomic_fetch and modify functions

<table>
<thead>
<tr>
<th>Key</th>
<th>Op</th>
<th>Computation</th>
<th>Key</th>
<th>Op</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>+</td>
<td>addition</td>
<td>and</td>
<td>&amp;</td>
<td>bitwise and</td>
</tr>
<tr>
<td>sub</td>
<td>-</td>
<td>subtraction</td>
<td>min</td>
<td>min</td>
<td>compute min</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td>bitwise inclusive or</td>
<td>max</td>
<td>max</td>
<td>compute max</td>
</tr>
<tr>
<td>xor</td>
<td>^</td>
<td>bitwise exclusive or</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Atomic Types and Enum Constants

memory_scope_sub_group requires the cl_khr_subgroups extension.

Parameter Type | Values
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>memory_order</td>
<td>memory_order_acq_rel memory_order_release</td>
</tr>
<tr>
<td>memory_scope</td>
<td>memory_order_acquire memory_order_seq_cst</td>
</tr>
<tr>
<td>memory_scope_work_group</td>
<td>memory_order_acquire memory_order_seq_cst</td>
</tr>
<tr>
<td>memory_scope_sub_group</td>
<td>memory_order_acquire memory_order_seq_cst</td>
</tr>
<tr>
<td>memory_scope_device</td>
<td>memory_order_acquire memory_order_seq_cst</td>
</tr>
<tr>
<td>memory_scope_work_item</td>
<td>memory_order_acquire memory_order_seq_cst</td>
</tr>
<tr>
<td>memory_scope_image</td>
<td>memory_order_acquire memory_order_seq_cst</td>
</tr>
</tbody>
</table>

Atomic Integer and floating-point types

† indicates types supported by a limited subset of atomic operations
‡ indicates size depends on whether implemented on 64-bit or 32-bit architecture
§ indicates types supported only if both 64-bit extensions are supported.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic_int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>atomic_uint</td>
<td>4 bytes</td>
</tr>
<tr>
<td>atomic_flag</td>
<td>1 byte</td>
</tr>
<tr>
<td>atomic_uchar</td>
<td>1 byte</td>
</tr>
<tr>
<td>atomic_ushort</td>
<td>2 bytes</td>
</tr>
<tr>
<td>atomic_ulong</td>
<td>8 bytes</td>
</tr>
<tr>
<td>atomic_ulonglong</td>
<td>16 bytes</td>
</tr>
</tbody>
</table>

Atomic Macros

#define ATOMIC_VAR_INIT(C) | Expands to a token sequence to initialize an atomic object of a type that is initialization-compatible with value. |
#define ATOMIC_FLAG_INIT | Initialize 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.

- `global T *` to `global (const T *)`
- `local T *` to `local (const T *)`
- `private T *` to `private (const T *)`
- `cl_mem_fence_flags` to `CL_KHR_GLOBAL_MEM_FENCE`

Miscellaneous Vector Functions [6.13.12]

- `uchar4 uc = (uchar4)(0xFA, 0xFB, 0xFC, 0xFD);`
- `void (^pipe)(T * p)`, `void (^pipe)(T * p, const T *)`, `void (^pipe)(T * p, T n mask)`, `void (^pipe)(T * p, T n mask, T m mask)`
- `void workgroup_commit_read_pipe(pipe T p, reserve_id_t reserve_id)`, `void workgroup_commit_write_pipe(pipe T p, reserve_id_t reserve_id)`
- `uint pipe_num_max_packets(pipe T p)`
- `uint pipe_num_packets(pipe T p)`

Pipe-Built-In Functions [6.13.16-2.4]

- `bool is_valid_reserve_id(reserve_id_t reserve_id)`
- `void commit_read_pipe(pipe T p, uint num_packets)`, `void commit_write_pipe(pipe T p, uint num_packets)`
- `void get_pipe_max_packets(pipe T p)`
- `uint get_pipe_num_packets(pipe T p)`

Workgroup Functions [6.13.15] [9.17.3.4]

- `int workgroup_all(int predicate)`
- `int workgroup_any(int predicate)`
- `int workgroup_all(uint predicate)`
- `int workgroup_any(uint predicate)`
- `int workgroup_all() (int predicate)`
- `int workgroup_any() (int predicate)`
- `int workgroup_all() (uint predicate)`
- `int workgroup_any() (uint predicate)`

Enqueuing and Kernel Query Built-In Functions [6.13.17] [8.17.3.6]

- `uint get_kernel_workgroup_size(void (*block)(void))`, `uint get_kernel_workgroup_size(void (*block) local void * ...)`
- `uint get_kernel_preferred_workgroup_size(void (*block)(void))`, `uint get_kernel_preferred_workgroup_size(void (*block) local void * ...)`
- `uint get_kernel_sub_group_count_for_nrange(const ndrange_t nrange, void (*block)(void))`, `uint get_kernel_sub_group_count_for_nrange(const ndrange_t nrange, void (*block) local void * ...)`
- `uint get_kernel_max_sub_group_count_for_nrange(const ndrange_t nrange, void (*block)(void))`, `uint get_kernel_max_sub_group_count_for_nrange(const ndrange_t nrange, void (*block) local void * ...)`

print Function [6.13.13]

Write output to an implementation-defined stream.

- `int 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.

print format string

The format string follows C99 conventions and supports an optional vector specifier:

%[flags][width][.precision][vector][length] conversion

Examples:

- `float4 A = (float4)(0.0f, 1.0f, 2.0f, 3.0f);`  
- `printf("A = \%.2f \%.2f \%.2f \%.2f\n", A.x, A.y, A.z, A.w);`

Enqueue example:

- `enqueue_kernel(queue_t queue, kernel enqueue flags, T * t_flag)`

Subgroups require the cl_khr_subgroups extension. Do an exclusive or inclusive scan operation specified by `<op>` of all values specified by work-items in the work-group or sub-group. The scan results are returned for each work-item. `<op>` may be min, max, or add.

- `workgroup broadcast([T n mask])`  
- `workgroup broadcast([T n mask], [T n mask])`

Subgroup broadcast requires the cl_khr_fp16 extension. Sub-group requires the cl_khr_subgroups extension. Double or vector types require double precision support.

- `sub_group_scan_exclusive<op>(int predicate)`
- `sub_group_scan_inclusive<op>(int predicate)`

Subgroups require the cl_khr_subgroups extension. Do an exclusive or inclusive scan operation specified by `<op>` of all values specified by work-items in the work-group or sub-group.

- `workgroup broadcast()`
- `sub_group_broadcast()`

Do an exclusive or inclusive scan operation specified by `<op>` of all values specified by work-items in the work-group or sub-group. The scan results are returned for each work-item. `<op>` may be min, max, or add.

- `workgroup broadcast([T n mask])`
- `workgroup broadcast([T n mask], [T n mask])`

Enqueue example:

- `enqueue_kernel(queue_t queue, kernel enqueue flags, T * t_flag)`

Subgroups require the cl_khr_subgroups extension. Do an exclusive or inclusive scan operation specified by `<op>` of all values specified by work-items in the work-group or sub-group.
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 Objects

In items in blue apply when the appropriate extension is supported.

Create Image Objects

cl_mem cAllocateImage(cl_context context, cl_mem_flags flags, const size_t *image_format, const size_t *image_desc, void *host_ptr, cl_int errcode_ret)
flags: See cCreateBuffer

Query List of Supported Image Formats

cl_int cGetSupportedImageFormats(cl_context context, cl_mem_flags flags, cl_mem_object_type image_type, cl_uchar num_entries, cl_image_format *image_formats)
flags: See cCreateBuffer

Read, Write, Copy, Fill Image Objects

cl_int cEnqueueReadImage(cl_command_queue command_queue, cl_mem image, cl_bool blocking_read, size_t origin, size_t region, size_t row_pitch, size_t slice_pitch, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
cl_int cEnqueueWriteImage(cl_command_queue command_queue, cl_mem image, cl_bool blocking_write, const size_t *origin, const size_t *region, size_t row_pitch, size_t slice_pitch, const void *ptr, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
cl_int cEnqueueFillImage(cl_command_queue command_queue, cl_mem image, cl_bool 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 cEnqueueCopyImage(cl_command_queue command_queue, cl_mem src_image, cl_mem dst_image, cl_mem src_buffer, cl_mem dst_buffer, size_t src_origin, size_t dst_origin, size_t src_region, size_t dst_region, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
cl_int cEnqueueCopyImageToBuffer(cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer, const size_t *src_origin, const size_t *src_region, size_t dst_offset, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)
cl_int cEnqueueCopyBufferToImage(cl_command_queue command_queue, cl_mem src_buffer, cl_mem dst_image, const size_t *src_offset, const size_t *dst_origin, const size_t *dst_region, cl_uint num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

Copy Between Image, Buffer Objects

Optional support [Table 5.6]

Additional notes:

- cl_khr_gl_depth_images extension is enabled.
- cAllocateImage uses the cl_khr_gl_depth_images extension.
- cAllocateImage uses the cl_khr_gl_depth_images extension.
- cAllocateImage uses the cl_khr_gl_depth_images extension.

Required channel data type = CL_UNORM_INT24 or CL_FLOAT.
The built-in functions defined in this section can only be used with image memory objects created with cDeviceImage. sampler specifies the addressing and filtering mode to use. cQual refers to one of the access qualifiers. For samplerless read functions this may be read_only or read_write.

### 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.

```c
void read_imagef(sampler sampler, coord coord, int2 coord, float4 color)
void write_imagef(sampler sampler, coord coord, int2 coord, float4 color)
void read_imageh(sampler sampler, coord coord, int2 coord, half4 color)
void write_imageh(sampler sampler, coord coord, int2 coord, half4 color)
```

Extended mipmap read and write functions [9.17.2.1]

These functions require the cl_khr_mipmap_image and cl_khr_mipmap_image_writes extensions.

```c
void read_image2d_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image2d_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image2d_array_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image2d_array_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image2d_array_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void write_image2d_array_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void read_image2d_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void write_image2d_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void read_image2d_buffer_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image2d_buffer_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image1d_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image1d_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image1d_array_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image1d_array_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image1d_array_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void write_image1d_array_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void read_image1d_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void write_image1d_t_image (sampler sampler, coord coord, int2 coord, uint4 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 [5.4].

```c
void read_image3d_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void write_image3d_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void read_image3d_array_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void write_image3d_array_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void read_image3d_buffer_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void write_image3d_buffer_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void read_image3d_depth_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void write_image3d_depth_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void read_image3d_array_depth_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void write_image3d_array_depth_t_image (sampler sampler, coord coord, int4 coord, float4 color)
void read_image3d_array_t_image (sampler sampler, coord coord, int4 coord, uint4 color)
void write_image3d_array_t_image (sampler sampler, coord coord, int4 coord, uint4 color)
void read_image3d_t_image (sampler sampler, coord coord, int4 coord, uint4 color)
void write_image3d_t_image (sampler sampler, coord coord, int4 coord, uint4 color)
```

Extended mipmap read and write functions [9.17.2.1]

These functions require the cl_khr_mipmap_image and cl_khr_mipmap_image_writes extensions.

```c
void read_image3d_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image3d_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image3d_array_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image3d_array_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image3d_array_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void write_image3d_array_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void read_image3d_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void write_image3d_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void read_image3d_buffer_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image3d_buffer_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image1d_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image1d_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image1d_array_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void write_image1d_array_depth_t_image (sampler sampler, coord coord, int2 coord, float4 color)
void read_image1d_array_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void write_image1d_array_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void read_image1d_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
void write_image1d_t_image (sampler sampler, coord coord, int2 coord, uint4 color)
```
Image Read and Write (continued)

Extended mipmap read and write functions (cont’d)

```c
int4 read_image4u(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod, float4 color)
int4 read_image4ui(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod, uint4 color)
void write_image4f(image2d_array_t image, int2 coord, int4 color)
void write_image4ui(image2d_array_t image, int2 coord, int4 color)
```


The MSAA forms require the extension cl_khr_gl_msaa_sharing. Mipmaps require the extension cl_khr_mipmap_image.

Query image width, height, and depth in pixels

```c
int get_image_width(image2d_t image)
int get_image_width(image2d_array_t image, sampler_t sampler)
int get_image_width(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
int get_image_height(image2d_t image)
int get_image_height(image2d_array_t image, sampler_t sampler)
int get_image_height(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
int get_image_depth(image3d_t image)
int get_image_depth(image3d_array_t image, sampler_t sampler)
int get_image_depth(image3d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
```

Query image array size

```c
size_t get_image_array_size(image2d_t image)
size_t get_image_array_size(image2d_array_t image, sampler_t sampler)
size_t get_image_array_size(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
size_t get_image2d_array_size(image2d_array_t image, sampler_t sampler)
size_t get_image2d_array_size(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
```

Query image dimensions

```c
int2 get_image_dim(image2d_t image)
int2 get_image_dim(image2d_array_t image, sampler_t sampler)
int2 get_image_dim(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
int2 get_image2d_dim(image2d_array_t image, sampler_t sampler)
int2 get_image2d_dim(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
```

Query image Channel data type and order

```c
int get_image_channel_data_type(image2d_t image, sampler_t sampler)
int get_image_channel_data_type(image2d_array_t image, sampler_t sampler)
int get_image_channel_data_type(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
```

Extended query functions [9.18.2.3]

These functions require the cl_khr_mipmap_image extension.

```c
int get_image_num_mip_levels(image2d_t image, sampler_t sampler)
int get_image_num_mip_levels(image2d_array_t image, sampler_t sampler)
int get_image_num_mip_levels(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
int get_image_num_samples(image2d_t image, sampler_t sampler)
int get_image_num_samples(image2d_array_t image, sampler_t sampler)
int get_image_num_samples(image2d_array_t image, sampler_t sampler, cl_int coord, cl_int lod)
```

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.

```c
__read_only, read only
__write_only, write only
```

Sampler Objects [5.7]

Items in blue require the cl_khr_mipmap_image extension.

```c
cl_sampler clCreateSamplerWithProperties(cl_context context, const cl_sampler_properties *sampler_properties, cl_int *errcode_ret)
sampler_properties: [Table 5.15]
CL_SAMPLER_NORMALIZED_COORDS,
CL_SAMPLER_ADDRESSING, FILTER_MODE
CL_SAMPLER_MIP_FILTER_MODE
CL_SAMPLER_LOD, [MIN, MAX]
```

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

```c
const sampler_t sampler_name = <normalized-mode> | <address-mode> | <filter-mode>
```


```
(CLK_NORMALIZED_COORDS | (CLK_ADDRESS | (CLK_MIP_FILTER | CLK_ADDRESSING | CLK_NORMALIZED_COORDS))
CLK_ADDRESS, X, where X may be NONE, REPEAT, CLAMP, CLAMP_TO_EDGE, MIRRORED_REPEAT
filter_mode: CLK_FILTER_NEAREST, CLK_FILTER_LINEAR
```
OpenCL 2.1 Reference Guide

OpenCL Extensions Reference

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 Extensions Reference

CL Image Objects > GL Renderbuffers [9.6.4]
cl_mem cCreateFromGLRenderbuffer (cl_context context, cl_mem_flags flags, GLuint renderbuffer, cl_int *errcode_ret)
flags: See cCreateFromGLBuffer

Query Information [9.6.5]
cl_int clGetGLObjectInfo (cl_mem memeobj, cl_gl_texture_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

CL Buffer Objects > GL Buffer Objects [8.6.2]
cl_mem cCreateFromGLBuffer (cl_context context, cl_mem_flags flags, GLuint bufferobj, cl_int *errcode_ret)
flags: See cCreateFromGLBuffer

CL Image Objects > GL Textures [9.6.3]
cl_mem cCreateFromGLTexture (cl_context context, cl_mem_flags flags, GLenum texture_target, GLint mipmaplevel, GLuint texture, cl_int *errcode_ret)
flags: See cCreateFromGLBuffer
texture_target: GL_TEXTURE_1D, 2D, 3D, BUFFER, RECTANGLE,
GL_TEXTURE_CUBE_MAP, POSITIVE (X, Y, Z),
GL_TEXTURE_CUBE_MAP_NEGATIVE (X, Y, Z),
GL_TEXTURE_2D_MULTISAMPLE_ARRAY (Requires extension cl_khr_gl_msaa_sharing)

CL Event Objects > GL Sync Objects [9.7.4]
cl_event cCreateEventFromGLSyncKHR (cl_context context, GLsync sync, cl_int *errcode_ret)
Requires the cl_khr_gl_msaa_sharing extension.

Direct3D 11 Sharing [9.10.7.3 - 9.10.7.6]
These functions require the cl_khr_d3d11_sharing extension. Associated header file is cl_khr_d3d11.h.
c_int clGetDeviceIDsFromD3D11KHR (cl_platform_id platform, cl_d3d11_device_id device_id, cl_int *errcode_ret)
flags: See cCreateFromD3D11BufferKHR

c_int cCreateFromD3D11BufferKHR (cl_context context, cl_mem_flags flags, ID3D11Buffer *resource, cl_int *errcode_ret)
flags: See cCreateFromD3D11BufferKHR

Direct3D 10 Sharing [9.8.7]
These functions require the cl_khr_d3d10_sharing extension. The associated header file is cl_khr_d3d10.h.
c_int clGetDeviceIDsFromD3D10KHR (cl_platform_id platform, cl_d3d10_device_id device_id, cl_int *errcode_ret)
flags: See cCreateFromD3D10BufferKHR

EGL Interoperability [9.18, 9.19]
Create CL Image Objects from EGL [9.18, 9.19]
These functions require the extension cl_khr_eegl_image.
c_int cCreateFromEGLImageKHR (cl_context context, CleglDisplayKHR display, CleglImageKHR image, cl_mem_flags flags, cl_eegl_image_propertiesKHR *properties, cl_int *errcode_ret)

Create CL Event Objects from EGL [9.18, 9.19]
This function requires the extension cl_khr_eegl_image.
c_int cCreateEventFromEGLSyncKHR (cl_context context, CleglSyncKHR sync, CleglDisplayKHR 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.

```c
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_arg1(a, (local int *)p, ...));,
                     local_mem_size);
   // 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(local_work_size, local_work_size);
   // enqueue dp_func_A
   enqueue_kernel(q,
                  CLK_ENQUEUE_FLAGS_WAIT_KERNEL, ndrange,
                  dp_func_A_blk);
   ...
}
```

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.

```c
kernel 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);
   }
   ...
}
```