OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other devices. 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/opencn.

**OpenCL API Reference**

### 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.

**Querying platform info & devices [API 4.1]**

- `clGetPlatformIDs` (cl_uint num_entries, cl_platform_id platforms[cl_uint *num_platforms])

- `clGetPlatformInfo` (cl_platform_id platform, cl_platform_info prop_name, size_t *prop_value_size, void *prop_value, size_t *prop_value_size_ret)

  `param_name`:
  - `CL_PLATFORM_VERSION`
  - `CL_PLATFORM_VENDOR`
  - `CL_PLATFORM_NAME`
  - `CL_PLATFORM_EXTENSIONS`
  - `CL_PLATFORM_EXTENSIONS_NAME_LENGTH`
  - `CL_PLATFORM_EXTENSIONS_WITH_VERSION`
  - `CL_PLATFORM_EXTENSIONS_WITH_VERSION_NAME_LENGTH`
  - `CL_PLATFORM_EXTENSIONS_WITH_VENDOR`
  - `CL_PLATFORM_EXTENSIONS_WITH_VENDOR_NAME_LENGTH`
  - `CL_PLATFORM_EXTENSIONS_WITH_VERSION_VENDOR`

- `clGetDeviceIDs` (cl_platform_id platform, cl_device_type device_type, cl_uint num_devices, cl_device_id devices[cl_uint *num_devices])

  `device_type`:
  - `CL_DEVICE_TYPE_ACCELERATOR`
  - `CL_DEVICE_TYPE_CPU`
  - `CL_DEVICE_TYPE_GPU`

- `clGetDeviceInfo` (cl_device_id device, cl_device_info prop_name, size_t *prop_value_size, void *prop_value, size_t *prop_value_size_ret)

  `param_name`:
  - `CL_DEVICE_NAME`
  - `CL_DEVICE_VENDOR`
  - `CL_DEVICE_VENDOR_ID`
  - `CL_DEVICE_EXTENSION_NAME_LENGTH`
  - `CL_DEVICE_PLATFORM`
  - `CL_DEVICE_PLATFORM_ID`
  - `CL_DEVICE_TYPE`
  - `CL_DEVICE_MAX_COMPUTE_UNITS`
  - `CL_DEVICE_MAX_MEM_ALLOC_SIZE`
  - `CL_DEVICE_MAX_MEM_ALLOC_HOST_PTR`
  - `CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE`
  - `CL_DEVICE_MAX_CONSTANT_SIZE`
  - `CL_DEVICE_MAX_MEM_BOBJECTS`
  - `CL_DEVICE_MAX_MEM_MAPiards`

- `clGetContextProperties` (cl_context context, size_t *num_entries, cl_context_properties properties[cl_uint *num_properties])

### OpenCL Runtime

API calls that manage OpenCL objects such as command-queues, memory objects, program objects, kernel objects for _kernel_function_ 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 [API 5.1]**

- `clCreateCommandQueue` (cl_context context, cl_device_id device, size_t *param_value)

  *properties*:
  - `CL_QUEUEweetedness`
  - `CL_QUEUE_NAME`
  - `CL_QUEUE_TYPE`
  - `CL_QUEUE_CONTEXT`
  - `CL_QUEUE_DEVICE`

- `clReleaseCommandQueue` (cl_command_queue command_queue)

### Contexts [API 4.4]

**clCreateContext**

```c
cl_context clCreateContext (const cl_context_properties *properties, cl_uint num_devices, const cl_device_id *devices, void *pfn_notify, cl_int errcode_ret)
```

**clReleaseContext**

```c
cl_int clReleaseContext (cl_context context)
```

**clGetContextInfo**

```c
cl_int clGetContextInfo (cl_context context, cl_context_info param_name, size_t *param_value_size, void *param_value, size_t *param_value_size_ret)
```

**Partitioning a device [API 4.3]**

- `clCreateCommandQueueWithProperties` (cl_context context, cl_device_id device, cl_int errcode_ret)

  *properties*:
  - `CL_QUEUEweetedness`
  - `CL_QUEUE_NAME`
  - `CL_QUEUE_TYPE`
  - `CL_QUEUE_CONTEXT`
  - `CL_QUEUE_DEVICE`

- `clReleaseCommandQueue` (cl_command_queue command_queue)

**Get CL extension function pointers [Ext 1.3]**

```c
void *clGetExtensionFunctionAddressForPlatform (cl_platform_id platform, const char *funcname)
```

©2020 Khronos Group - Rev. 1020

www.khronos.org/opencn
Buffer Objects [API 5.2]
Elements of buffer objects are stored sequentially and accessed using a pointer by a kernel executing on a device.

Create buffer objects
cl_mem clCreateBuffer(  
    cl_context context, cl_mem_flags flags, size_t size,  
    void *host_ptr, cl_int *errcode_ret)  
flags: CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY,  
CL_MEM_HOST_NO_ACCESS, CL_MEM_HOST_READ_WRITE, ONLY,  
CL_MEM_USE_ALLOC, COPY_HOST_PTR

cl_mem clCreateBufferWithProperties (  
    cl_context context, const cl_mem_properties *properties, cl_mem_flags flags,  
    size_t size, void *host_ptr, cl_int *errcode_ret)  
flags: See clCreateBuffer

cl_mem clCreateSubBuffer (  
    cl_mem buffer, cl_mem_flags flags, cl_buffer_create_type buffer_type,  
    const void *buffer_create_info, cl_int *errcode_ret)  
flags: See clCreateBuffer

Read, write, copy, & fill buffer objects
cl_int clEnqueueReadBuffer (  
    cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read,  
    size_t offset, size_t size, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)  
cl_int clEnqueueReadBufferRect (  
    cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read,  
    size_t *buffer_origin, const size_t *host_origin, const size_t *region,  
    size_t *row_pitch, size_t *slice_pitch, size_t *host_row_pitch,  
    size_t *host_slice_pitch, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

c_int clEnqueueFillBuffer (  
    cl_command_queue command_queue, cl_mem buffer, const void *pattern,  
    const size_t *pattern_size, const size_t *size, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyBuffer (  
    cl_command_queue command_queue, cl_mem src_buffer, cl_mem dst_buffer,  
    size_t *src_offset, size_t *dst_offset, size_t *size, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

c_int clEnqueueCopyBufferRect (  
    cl_command_queue command_queue, cl_mem src_buffer, cl_mem dst_buffer,  
    const size_t *src_origin, const size_t *dst_origin, const size_t *region,  
    size_t *src_row_pitch, size_t *src_slice_pitch, size_t *dst_row_pitch,  
    size_t *dst_slice_pitch, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

Map buffer objects
void *clEnqueueMapBuffer (  
    cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_map,  
    cl_map_flags map_flags, size_t *size, size_t *offset, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event,  
    cl_int *errcode_ret)  
map_flags: CL_MAP_READ, CL_MAP_WRITE

cl_int clEnqueueWriteImage (  
    cl_command_queue command_queue, cl_mem image, cl_bool blocking_write,  
    size_t *origin, const size_t *region, size_t *size, ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

c_int clEnqueueWriteBuffer (  
    cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write,  
    size_t *offset, size_t *size, const void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueWriteBufferRect (  
    cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write,  
    const size_t *buffer_origin, const size_t *host_origin, const size_t *region,  
    size_t *row_pitch, size_t *slice_pitch, size_t *host_row_pitch,  
    size_t *host_slice_pitch, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueFillBuffer (  
    cl_command_queue command_queue, cl_mem buffer, const void *pattern,  
    const size_t *pattern_size, const size_t *size, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyBuffer (  
    cl_command_queue command_queue, cl_mem src_buffer, cl_mem dst_buffer,  
    size_t *src_offset, size_t *dst_offset, size_t *size, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

cl_int clEnqueueCopyBufferRect (  
    cl_command_queue command_queue, cl_mem src_buffer, cl_mem dst_buffer,  
    const size_t *src_origin, const size_t *dst_origin, const size_t *region,  
    size_t *src_row_pitch, size_t *src_slice_pitch, size_t *dst_row_pitch,  
    size_t *dst_slice_pitch, void *ptr, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event)

Copy between image & buffer objects
cl_int clEnqueueCopyImageToBuffer (  
    cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer,  
    size_t *src_offset, size_t *dst_offset, size_t *size, cl_int 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_int num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

Map and unmap image objects
void *clEnqueueMapImage (  
    cl_command_queue command_queue, cl_mem image, cl_bool blocking_map,  
    cl_map_flags map_flags, const size_t *origin, const size_t *region,  
    size_t *image_row_pitch, size_t *image_slice_pitch, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret)
map_flags: CL_MAP_READ, CL_MAP_WRITE

Copy between image & buffer objects
cl_int clEnqueueCopyImageToBuffer (  
    cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer,  
    size_t *src_offset, size_t *dst_offset, size_t *size, cl_int 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_int num_events_in_wait_list, cl_event *event_wait_list, cl_event *event)

Map and unmap image objects
void *clEnqueueMapImage (  
    cl_command_queue command_queue, cl_mem image, cl_bool blocking_map,  
    cl_map_flags map_flags, const size_t *origin, const size_t *region,  
    size_t *image_row_pitch, size_t *image_slice_pitch, cl_int num_events_in_wait_list,  
    cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret)
map_flags: CL_MAP_READ, CL_MAP_WRITE

Query image objects
cl_int clGetImageInfo (  
    cl_mem image, cl_image_info param_name, size_t param_value_size,  
    void *param_value, size_t *param_value_size_ret)  
param_name: CL_IMAGE_FORMAT, CL_IMAGE_FORMAT_SIZE,  
CL_IMAGE_SIZE, CL_IMAGE_ROW, CL_IMAGE_SLICE, CL_IMAGE_ARRAY_ELEMENT_SIZE,  
CL_IMAGE_MIP_HEIGHT, CL_IMAGE_MIP_WIDTH, CL_IMAGE_MIP_DEPTH)
## OpenCL 3.0 Reference Guide

### Pipes [API 5.4]
A pipe is a memory object that stores data organized as a FIFO. Pipes 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
- `cl_mem clCreatePipe (cl_context context, cl_mem_flags flags, cl_uint pipe_max_packets, const cl_pipe_properties *properties, cl_int *errcode_ret)`
  - `flags`: 0 or CL_MEM_READ_WRITE, CL_MEM_HOST_NO_ACCESS

### Shared Virtual Memory [API 5.6]
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.

#### Allocate and free SVM
- `void *clSVMAlloc (cl_context context, cl_mem_flags flags, size_t size, cl_uint alignment)`
- `cl_int clSVMFree (cl_context context, void *svm_pointer)`

#### SVM operations
- `cl_int clEnqueueSVMFree (cl_command_queue command_queue, cl_uint num_svm_pointers, const void *svm_pointers[], cl_int *errcode_ret)`
- `cl_int clEnqueueSVMMap (cl_command_queue command_queue, cl_buffer block_size, void *dst_ptr, const void *src_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`
- `cl_int clEnqueueSVMMemCopy (cl_command_queue command_queue, cl_buffer block_size, void *dst_ptr, const void *src_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`
- `cl_int clEnqueueSVMMemFill (cl_command_queue command_queue, cl_buffer block_size, void *dst_ptr, const void *src_pattern, size_t dst_pattern_size, size_t size, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`
- `cl_int clEnqueueSVMMap (cl_command_queue command_queue, cl_buffer block_size, void *dst_ptr, const void *src_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`
- `cl_int clEnqueueSVMUnmap (cl_command_queue command_queue, void *svm_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`
- `cl_int clEnqueueSVMMapMem (cl_command_queue command_queue, cl_buffer block_size, void *dst_ptr, cl_mem_map_flags map_flags, void *svm_ptr, size_t size, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`
- `cl_int clEnqueueSVMMapMemFill (cl_command_queue command_queue, cl_buffer block_size, void *dst_ptr, cl_mem_map_flags map_flags, const void *svm_pointers[], size_t size, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`
- `cl_int clEnqueueSVMMapMigrateMem (cl_command_queue command_queue, void *svm_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

### Memory Objects [API 5.5]
A memory object is a handle to a reference counted region of global memory. Includes buffer objects, image objects, and global memory. Includes buffer objects, image objects, and global memory.

#### Memory objects
- `cl_mem clRetainMemObject (cl_mem memobj)`
- `cl_mem clReleaseMemObject (cl_mem memobj)`
- `cl_int clSetMemObjectDestructorCallback (cl_mem memobj, void (CL_CALLBACK *pfn_free_func), void *user_data)`
- `cl_int clEnqueueSVMMap (cl_command_queue command_queue, cl_mem memobj, void *mapped_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`
- `cl_int clEnqueueSVMUnmapMemObject (cl_command_queue command_queue, cl_mem memobj, void *unmapped_ptr, void *user_data)`
- `cl_int clGetMemObjectInfo (cl_mem memobj, const cl_mem_info *param_name, size_t *param_value_size, void **param_value, size_t *param_value_size_ret, void *user_data)`

### Sampler Objects [API 5.7]
Samplers are used for texture mapping and provide functions that map 1D, 2D, and 3D texture coordinates to memory addresses. They are used for accessing textures in kernels.

#### Sampler Objects
- `cl_sampler clCreateSamplerWithProperties (cl_context context, const cl_sampler_properties *properties, cl_int *errcode_ret)`
- `cl_sampler clCreateSampler (cl_context context)`
- `cl_sampler clCreateSamplerWithParameters (cl_context context, const cl_sampler_properties *parameters, cl_int *errcode_ret)`
- `cl_int clSetKernelArg (cl_program program, size_t param_name, size_t *param_value_size, const void *param_value, size_t *param_value_size_ret)`

### Program Objects [API 5.8]
OpenCL programs consist of sets of kernels identified as functions declared with the `__kernel` qualifier in the program source.

#### Program Objects
- `cl_program clCreateProgramWithSource (cl_context context, const char *source, size_t length, cl_uint *errcode_ret)`
- `cl_program clCreateProgramWithBinary (cl_context context, cl_mem binary, cl_int *errcode_ret)`
- `cl_program clCreateProgramWithClSource (cl_context context, cl_int source_size, void *source, cl_int *errcode_ret)`
- `cl_program clCreateProgramWithClSources (cl_context context, cl_int num_input_programs, const cl_device_id *device_list, const char *input_headers[], cl_int *errcode_ret)`
- `cl_program clGetProgramInfo (cl_program program, const char *param_name, void *param_value, size_t *param_value_size)`

### Migrate memory objects
- `cl_int clEnqueueMigrateMemObjects (cl_command_queue command_queue, cl_int num_mem_objects, const cl_mem *mem_objects, cl_mem_migration_flags flags, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

### Memory operations
- `cl_int clGetMemObjectInfo (cl_mem memobj, const cl_mem_info *param_name, size_t *param_value_size, void *param_value)`

### Query memory object
- `cl_int clGetMemObjectInfo (cl_mem memobj, const cl_mem_info *param_name, size_t *param_value_size, void *param_value)`

### Sampler declaration fields
<table>
<thead>
<tr>
<th>Sampler declaration fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampler declaration fields</td>
</tr>
</tbody>
</table>

### Separate compilation and linking
- `cl_int clCompileProgram (cl_context context, cl_int num_devices, const cl_device_id *device_list, const char *options, const cl_uint *num_input_programs, const char **input_headers, cl_int *errcode_ret)`
- `cl_int clLinkProgram (cl_context context, cl_int num_devices, const cl_device_id *device_list, const char *options, const cl_uint *num_input_programs, const cl_mem *input_programs, void (CL_CALLBACK *pfn_notify), cl_int *errcode_ret)`

### Unlock the OpenCL compiler
- `cl_int clUnlockPlatformCompiler (cl_platform_id platform)`

### 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)`

### Page 3
©2020 Khronos Group - Rev. 1020

(Continued on next page >)
**Program Objects (continued)**

**Compiler options**

**Preprocessor**
- `-D` processed in order for `clBuildProgram` or `clCompileProgram`
- `-D` name=definition -I dir

**Math intrinsics**:
- `-single-precision-constant`
- `-denorms-are-zero`
- `-fp32-correctly-rounded-divide-sqrt`
- `-fstrict-aliasing`

**OpenCL 3.0 Reference Guide**

Granularity of sharing

**clSVMAlloc**

Host synchronization points between

No

Yes, through `Map` and `Unmap` commands.

Memory allocation

- `cl_event`
- `cl_int`
- `cl_mem`

Mechanisms to enforce consistency

- `cl_event`
- `cl_int`
- `cl_mem`

**Warning request/suppress**

- `-Werror`

Control OpenCL Language version

- `-std=CL1.1` OpenCL 1.1 specification
- `-std=CL1.2` OpenCL 1.2 specification
- `-std=CL2.0` OpenCL 2.0 specification
- `-std=CL3.0` OpenCL 3.0 specification

Query kernel argument information:

- `cl_kernel-arg-info`

**Optimization options**

- `-cl-opt-disable`
- `-cl-no-signed-zeros`
- `-cl-unsafe-math-optimizations`
- `-cl-uniform-work-group-size`
- `-cl-parallel-work-group-size`
- `-cl-mad-enable`
- `-cl-fast-math-only`
- `-cl-fast-relaxed-math`
- `-cl-no-subgroup-ifp`

**Debugging options**

- `-g`

- Generate additional errors for built-in functions that allow you to enqueue commands on a device

**Linker options**

**Library linking options**

- `-enable-link-options`

**Program linking options**

- `-cl-denoms-are-zero`
- `-cl-no-signed-zeros`
- `-cl-finite-math-only`
- `-cl-fp32-correctly-rounded-divide-sqrt`
- `-cl-unsafe-math-optimizations`

**Kernel Objects [API 5.9 - 5.10]**

A kernel object encapsulates the specific __kernel function and the argument values to be used when executing it.

**Create kernel objects**

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

**Kernel arguments and queries**

- `cl_int clSetKernelArg (cl_kernel kernel, cl_int arg_index, size_t arg_size, const void * arg_value)`
- `cl_int clSetKernelArgSVMPointer (cl_kernel kernel, cl_int arg_index, size_t arg_size, const void * arg_value, size_t param_value_size, const size_t * param_value_size_ret)`
- `cl_int clGetKernelArgInfo (cl_kernel kernel, cl_int arg_index, cl_kernel_arg_info * info, size_t * param_value_size, const size_t * param_value_size_ret)`
- `cl_int clSetKernelExecInfo (cl_kernel kernel, cl_kernel_exec_info param_name, const size_t * param_value_size, void * param_value, size_t * param_value_size_ret)`
- `cl_int clGetKernelExecInfo (cl_kernel kernel, cl_kernel_exec_info param_name, void * param_value, size_t * param_value_size, const void * param_value, size_t * param_value_size_ret)`

**Event Objects [API 5.11 - 5.14]**

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**

- `cl_event clCreateUserEvent (cl_context context, cl_int errcode_ret)`
- `cl_int clSetUserEventStatus (cl_event event, cl_int execution_status)`
- `cl_int clWaitForEvents (cl_uint num_events, const cl_event * event_list)`
- `cl_int clGetEventInfo (cl_event event, cl_event_info_info param_name, size_t param_value_size, void * param_value, size_t * param_value_size_ret)`

**Markers, barriers, & waiting for events**

- `cl_int clEnqueueMarkerWithWaitList (cl_command_queue command_queue, cl_uint num_events_in_wait_list, const cl_event * event_wait_list, cl_event * event)`
- `cl_int clEnqueueBarrierWithWaitList (cl_command_queue command_queue, cl_uint num_events_in_wait_list, const cl_event * event_wait_list, cl_event * event)`

**Memory Model: SVM [API 3.3.3]**

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

- **Coarse-grained buffer SVM**: Sharing at the granularity of regions of OpenCL buffer memory objects.
- **Fine-grained buffer SVM**: Sharing occurs at the granularity of individual loads/stores into bytes within OpenCL buffer memory objects.
- **Fine-grained system SVM**: Sharing occurs at the granularity of individual loads/stores into bytes occurring anywhere within the host memory.

**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</td>
<td><code>clCreateBuffer</code></td>
<td>Host synchronization points on the same or between devices.</td>
<td>Yes, through Map and Unmap commands.</td>
</tr>
<tr>
<td></td>
<td>(buffer)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coarse-grained buffer SVM</td>
<td>OpenCL Memory objects</td>
<td><code>clSVMAlloc</code></td>
<td>Host synchronization points between devices.</td>
<td>Yes, through Map and Unmap commands.</td>
</tr>
<tr>
<td></td>
<td>(buffer)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fine Grained buffer SVM</td>
<td>Bytes within OpenCL Memory objects (buffer)</td>
<td><code>clSVMAlloc</code></td>
<td>Synchronization points plus atoms if supported</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 atoms if supported</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Supported Data Types

**Built-in Scalar Data Types**

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>cl_bool</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</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</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</td>
<td>cl_uint</td>
<td>32-bit unsigned</td>
</tr>
<tr>
<td>long</td>
<td>cl_long</td>
<td>32-bit signed or 64-bit unsigned</td>
</tr>
<tr>
<td>unsigned long</td>
<td>cl_ulong</td>
<td>32-bit signed or 64-bit unsigned</td>
</tr>
</tbody>
</table>

**Built-in Vector Data Types**

- Substitutes integer 3.0 for 3.0
- v.s3, v.s5, v.s13
- v.s5, v.s8
- 5
- 14
- v.s4, v.s13
- 5
- 14
- v.s0, v.s2, v.s4, v.s02
- cl_float
- Integer line number
- 1 if device is little endian
- 32- or 64-bit unsigned integer
- v.x, v.s0
- float8
- v.hi
- true (1) or false (0)
- cl_[u]int
- 32-bit unsigned / 32-bit float
- [C 6.3]

**Other Built-in Data Types**

<table>
<thead>
<tr>
<th>OpenCL Type</th>
<th>API Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 float</td>
</tr>
<tr>
<td>half</td>
<td>cl_half</td>
<td>16-bit float</td>
</tr>
<tr>
<td>size_t</td>
<td>--</td>
<td>32- or 64-bit unsigned integer</td>
</tr>
<tr>
<td>ptdiff_t</td>
<td>--</td>
<td>32- or 64-bit signed integer</td>
</tr>
<tr>
<td>intrptr_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>

### Vector Component Addressing

**Vector Components**

- v.x, v.y, v.z
- v.s0, v.s1, v.s2
- v.xx, v.xy, v.zz
- void

**Vector Addressing Equivalences**

- Numeric indices are preceded by the letter s or S, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: vxy, vx, v.xy.

<table>
<thead>
<tr>
<th>v.lo</th>
<th>v.s0</th>
<th>v.s1</th>
<th>v.s2</th>
<th>v.xx</th>
<th>v.xy</th>
<th>v.zz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Preprocessor Directives & Macros**

- #pragma OPENCL_FP_CONTRACT
- #pragma OPENCL_EXTENSION
- #pragma OPENCL_VERSION

**Conversions, Type Casting Examples**

- T a = (T) b; // Scalar to scalar, or scalar to vector
- T a = convert_f_i(T b);  // cast from float to int
- T a = as(T b);  // cast to nearest even
- T a = convert_sat_f_i(T b);  // convert to signed and saturate

### Operators and Qualifiers

**Operators**

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

**Address Space Qualifiers**

- __global__, __local__, __constant__, __private__, __shared__

**Function Qualifiers**

- __kernel__, __event__, __device__, __function__, __global__, __local__, __constant__, __private__, __shared__

**Attribute Qualifiers**

- __FILE__, __LINE__, __OPENCL_VERSION__, __CL_VERSION__, __CL_VERSION__, __OPENCL_C_VERSION__, __ENDIAN_BIG__, __ENDIAN_LITTLE__

- #pragma OPENCL_FP_CONTRACT
- #pragma OPENCL_EXTENSION

- T a = (T) b; // Scalar to scalar, or scalar to vector
- T a = convert_f_i(T b);  // cast from float to int
- T a = as(T b);  // cast to nearest even
- T a = convert_sat_f_i(T b);  // convert to signed and saturate

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

- __attribute__((aligned(n)))
- __attribute__((packed))

Use to specify attributes of variables or structure fields.

- __attribute__((vec_type_hint(type)))
- __attribute__((vec_type_hint(type)))

Use to specify basic blocks and control-flow statements.

- __attribute__((onoffswitch))...

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

- __attribute__((opencl_unroll_hint(n)))
- __attribute__((opencl_unroll_hint(n)))

©2020 Khronos Group - Rev. 1020

www.khronos.org/opencl
Blocks [C.6.12]

A result value type with a list of parameter types. Requires support for the __global, __local, or __private, or may be the generic address space with the __global, __local, or __private, or a combination of the two.

Number of work-items in the work-group

Sub-group ID

Number of work-items in the work-group

Work-item global ID

Global offset

### Math Built-in Functions [C.6.13.2]

The type used in a function must be the same for all arguments and the return type unless otherwise specified.

### Math Constants [C.6.13.3]

The values of the following symbolic constants are single-precision float.

**MAXFLOAT**

Value of maximum non-infinite single-precision floating-point number

**HUGE_VALF**

Positive float expression, evaluates to +infinity

**INFINITY**

Constant float expression, positive or unsigned infinity

**NAN**

Constant float expression, quiet NaN

**HUGE_VAL**

Positive double expression, evals. to +infinity (Requires double precision support.)

When double precision is supported, macros ending in _F are available in type double by removing _F from the macro name.

**M_E**

Value of e

**M_LOG2E**

Value of log2_e

**M_LOG10E**

Value of log10_e

**M_LN2**

Value of log2

**M_LN10**

Value of log10

**M_PI**

Value of π

**M_PI_2**

Value of π/2

**M_PI_4**

Value of π/4

**M_1_PI**

Value of 1/π

**M_2_PI**

Value of 2/π

**M_SQRTPI**

Value of 2√π

**M_SQRT2**

Value of 2√2
**Image Read and Write Functions** [C 6.13.14]

The built-in functions defined in this section can only be used with image memory objects created with clCreateImage. sampler specifies the addressing and filtering mode to use. aQual 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
float4 read_imagef(read_only image2d_t image, sampler_t sampler, int2 coord);
int4 read_imagei(read_only image2d_t image, sampler_t sampler, int2 coord);
uint4 read_imageui(read_only image2d_t image, sampler_t sampler, int2 coord);
float4 read_imagef(image2d_t image, int2 coord, float depth);
int4 read_imagei(image2d_t image, int2 coord, float depth);
uint4 read_imageui(image2d_t image, int2 coord, float depth);
```

**Image Query Functions** [C 6.13.14]

**Query image dimensions**

```c
int2 get_image_dim(image2d_t image, int2 coord, int4 color);
```

**Query image channel data type and order**

```c
void write_imagef(image2d_t image, int coord, float4 color);
void write_imagei(image2d_t image, int coord, int4 color);
void write_imageui(image2d_t image, int coord, uint4 color);
void write_imagef(image3d_t image, int coord, float4 color);
void write_imagei(image3d_t image, int coord, int4 color);
void write_imageui(image3d_t image, int coord, uint4 color);
void write_imagef(buffer_t image, int coord, float4 color);
void write_imagei(buffer_t image, int coord, int4 color);
void write_imageui(buffer_t image, int coord, uint4 color);
```

**Common Built-in Functions** [C 6.13.14]

These functions operate component-wise and use round to nearest even rounding mode. Ts is type float. If supported, Ts can also be type double. Tn is the vector form of Ts, where n is 2, 3, 4, 8, or 16. T is type Ts and Tn.

```c
int clamp(T x, T min, T max);
float clamp(T x, T min, T max);
```

```c
T degrees(T radians);
T radians(T degrees);
```

```c
T max(T x, T y);
T min(T x, T y);
```

```c
T mix(T x, T y, T a);
T mix(T x, T y, T a);
```

```c
T step(T x, T y);
T step(T x, T y);
```

```c
T smoothstep(T x, T y, T x);
T smoothstep(T x, T y, T x);
```

```c
T sign(T x);
T sign(T x);
```
Integer Built-in Functions [C 6.13.3]

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs(T t)</code></td>
<td></td>
</tr>
<tr>
<td><code>abs_diff(T t, T y)</code></td>
<td></td>
</tr>
<tr>
<td><code>add_sat(T x, T y)</code></td>
<td>x + y and saturates the result</td>
</tr>
<tr>
<td><code>addh(T x, T y)</code></td>
<td>(x + y) &gt;&gt;</td>
</tr>
<tr>
<td><code>clamp(T x, T min, T max)</code></td>
<td>min(x, min), max(x), maxval</td>
</tr>
<tr>
<td><code>cld(T x)</code></td>
<td>Number of leading 0-bits in x</td>
</tr>
<tr>
<td><code>cld(T x)</code></td>
<td>Number of trailing 0-bits in x</td>
</tr>
<tr>
<td><code>madhi(T a, T b, T c)</code></td>
<td>mul_hi(x, c) + b</td>
</tr>
<tr>
<td><code>mad_sat(T x, T y)</code></td>
<td>o * b + c and saturates the result</td>
</tr>
<tr>
<td><code>max(T x, T y)</code></td>
<td>y if x &lt; y, else returns x</td>
</tr>
<tr>
<td><code>max(T x, T y)</code></td>
<td>y if x &lt; y, else returns x</td>
</tr>
<tr>
<td><code>mul_hi(T x, T y)</code></td>
<td>High half of the product of x and x</td>
</tr>
<tr>
<td><code>rotate(T x, T y)</code></td>
<td>result[idx] = x[idx] &lt;&lt; (idx)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sub_sat(T x, T y)</code></td>
<td>x - y and saturates the result</td>
</tr>
<tr>
<td><code>popcount(T x)</code></td>
<td>Number of non-zero bits in x</td>
</tr>
</tbody>
</table>

Relational Built-in Functions [C 6.13.6]

These functions can be used with built-in scalar or vector types as arguments and return a scalar or vector integer result.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isequal(float x, float y)</code></td>
<td>Compare of x == y</td>
</tr>
<tr>
<td><code>insequal(double x, double y)</code></td>
<td>Compare of x == y</td>
</tr>
<tr>
<td><code>isnotequal(float x, float y)</code></td>
<td>Compare of x != y</td>
</tr>
<tr>
<td><code>isnotequal(double x, double y)</code></td>
<td>Compare of x != y</td>
</tr>
<tr>
<td><code>isgreater(float x, float y)</code></td>
<td>Compare of x &gt; y</td>
</tr>
<tr>
<td><code>isgreater(double x, double y)</code></td>
<td>Compare of x &gt; y</td>
</tr>
<tr>
<td><code>isgreaterequal(float x, float y)</code></td>
<td>Compare of x &gt;= y</td>
</tr>
<tr>
<td><code>isgreaterequal(double x, double y)</code></td>
<td>Compare of x &gt;= y</td>
</tr>
<tr>
<td><code>islessequal(float x, float y)</code></td>
<td>Compare of (x &lt; y)</td>
</tr>
<tr>
<td><code>islessequal(double x, double y)</code></td>
<td>Compare of (x &lt;= y)</td>
</tr>
<tr>
<td><code>isfinite(float x)</code></td>
<td>Test for finite value</td>
</tr>
<tr>
<td><code>isfinite(double x)</code></td>
<td>Test for finite value</td>
</tr>
<tr>
<td><code>isinf(float x)</code></td>
<td>Test for + or - infinity</td>
</tr>
<tr>
<td><code>isinf(double x)</code></td>
<td>Test for + or - infinity</td>
</tr>
<tr>
<td><code>isnan(float x)</code></td>
<td>Test for NaN</td>
</tr>
<tr>
<td><code>isnan(double x)</code></td>
<td>Test for NaN</td>
</tr>
<tr>
<td><code>iscnormal(float x)</code></td>
<td>Test for a normal value</td>
</tr>
<tr>
<td><code>iscnormal(double x)</code></td>
<td>Test for a normal value</td>
</tr>
</tbody>
</table>

Vector Data Load/Store [C 6.13.7]

These functions can be used with built-in scalar or vector types as arguments and return a scalar or vector integer result.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vloada_half(size_t offset, const [constant] * t)</code></td>
<td>Read half vector data from address (o + offset * 2)</td>
</tr>
<tr>
<td><code>vstorea_half(size_t offset, const [constant] half * t)</code></td>
<td>Write a half to address (o + offset)</td>
</tr>
</tbody>
</table>

For each component of a vector type, result[i] = if MSB of c[i] is set ?1 : 0; 0 |

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vload(a, T b, T c)</code></td>
<td>Each bit of result is corresponding bit of a if corresponding bit of c is 0</td>
</tr>
<tr>
<td><code>vstorea_half(size_t offset, const [constant] half * t)</code></td>
<td>Write a half to address (o + offset)</td>
</tr>
</tbody>
</table>

For half3, read from aligned (o + offset * 3). | Read half vector data from aligned (o + offset * 3). |

void vstorea_half(size_t offset, const [constant] half * t) |

Write a half vector data to | Write a half vector to address (o + offset) |

float vloada_half(size_t offset, const [constant] half * t) |

Read half vector data from | Read a half from address (o + offset) |

void vstorea_half(size_t offset, const [constant] half * t) |

Write a half to address (o + offset) |

float vload(a, T b, T c) |

Each bit of result is corresponding bit of a if corresponding bit of c is 0; 0 | For each component of a vector type, result[i] = if MSB of c[i] is set ?1 : 0; 0; For scalar type, result = c ? b : a | Read half vector data from aligned (o + offset * 3). | Read a half from address (o + offset) |

void vstorea_half(size_t offset, const [constant] half * t) |

Write a half vector data to | Write a half vector to address (o + offset) |

float vstorea_half(size_t offset, const [constant] half * t) |

Write a half to address (o + offset) |

float vstorea_half(size_t offset, const [constant] half * t) |

Write a half vector data to | Write a half vector to address (o + offset) |
Synchronization & Memory Fence Functions [C 6.13.8]

flags argument is the memory address space, set to a 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.

```
void barrier(cl_mem_fence_flags flags)
void work_group_barrier(cl_mem_fence_flags flags [], memory_scope work_item)
void sub_group_barrier(cl_mem_fence_flags flags [], memory_scope work Item in a sub-group must execute this before any can continue.
```

Async Copies and Prefetch [C 6.13.10]

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

```
void atomic_flag_test_and_set_explicit(void atomic_flag *object, memory_order order, memory_scope scope)
```

Miscellaneous Vector Functions [C 6.13.12]

Tn and Tm are type char, uchar, short, ushort, int, uint, shortn, ulong, or double
where n is 2, 4, 8, or 16 except in vec_step it may also be 3. Tn is uchar, ushort, uint, or ulong.
In all types listed here, long, ulong, or double available only if supported.

```
int vec_step(Tn a) Takes built-in scalar or vector data type argument. Returns 1 for scalar,
int vec_step (type) 4 for 3-component vector, else number of elements in
Tn a [] memory_scope
```

Atomic Functions [C 6.13.11]

OpenCL C 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 ptrdiff_t.

The atomic_double, atomic_long, and atomic_ulong types are available if supported. The default
scope is memory_scope_work_group for local atoms and memory_scope_device for global
atomics. The default scope is memory_scope_work_group for local atoms and memory_scope_device
device for global atomics, therefore the non-explicit functions require OpenCL C 2.0 or both the features
__opencl_c_atomic_order_seq_cst and __opencl_c_atomic_device scope.

```
void atomic_init(volatile A *obj, C value)
void atomic_work_item_fence( cl_mem_fence_flags flags, memory_order order, memory_scope scope)
```

The atomic object pointer supports the global and local address spaces. The expected pointer
pointers supports the global, local, and private address spaces. For both pointers, the generic
address space supported is the __opencl_c_generic_address_space feature.

```
void atomic_store(volatile A *object, C desired) Atomically replace the value pointed to by
void atomic_store_explicit(volatile A *object, C desired, memory_order order, memory_scope scope) object with the value of desired. Memory is affected according to the value of order.
```

```
C atomic_load(volatile A *object, memory_order order, memory_scope scope) Atomically returns the value pointed to by object. Memory is affected according to the value of order.
```

```
C atomic_exchange(volatile A *object, C desired, memory_order order, memory_scope scope) Atomically replace the value pointed to by object with desired. Memory is affected according to the value of order.
```

```
bool atomic_compare_exchange_strong(volatile A *object, C expected, C desired, memory_order order, memory_scope scope) 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.
```

```
C atomic_fetch_add(volatile A *object, M operand)
C atomic_fetch_sub(volatile A *object, M operand)
```

```
bool atomic_flag_test_and_set(void atomic_flag *object, memory_order order, memory_scope scope) 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(void atomic_flag *object, memory_order order, memory_scope scope) Atomically sets the value pointed to by object to false. The order argument shall not be memory_order_acquire nor memory_order_acq_rel. Memory is affected according to the value of order.
```

Values for key for atomic_fetch and modify functions

```
key op computation
add + addition
sub - subtraction
or | bitwise inclusive or
xor ^ bitwise exclusive or
```

```
key op computation
and & bitwise and
& bitwise and
min min compute min
max max compute max
```

Atomic Types and Enum Constants

Parameter type: memory_order

```
<table>
<thead>
<tr>
<th>Values for memory_order</th>
<th>Optional requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory_order_relaxed</td>
<td>With any built-in atomic function except</td>
</tr>
<tr>
<td>memory_order_acquire</td>
<td>__atomic_work_item_fence, requires OpenCL C 2.0 or support for</td>
</tr>
<tr>
<td>memory_order_release</td>
<td>the __opencl_c_atomic_order_acquire feature</td>
</tr>
<tr>
<td>memory_order_acq_rel</td>
<td>Requires OpenCL C 2.0 or support for the</td>
</tr>
<tr>
<td>memory_order_seq_cst</td>
<td>__opencl_c_atomic_order_seq_cst feature</td>
</tr>
</tbody>
</table>
```

```
Parameter type: memory_scope

<table>
<thead>
<tr>
<th>Values for memory_scope</th>
<th>Optional requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory_scope_work_group</td>
<td>Only used with __atomic_work_item_fence with flags:</td>
</tr>
<tr>
<td>memory_scope_item</td>
<td>CLK_IMAGE_MEM_FENCE</td>
</tr>
<tr>
<td>memory_scope_sub_group</td>
<td>Requires support for the __opencl_c_subgroup feature</td>
</tr>
<tr>
<td>memory_scope_device</td>
<td>Requires OpenCL C 2.0 or support for the</td>
</tr>
<tr>
<td>memory_scope_all_svm_devices</td>
<td>__opencl_c_atomic_device feature</td>
</tr>
</tbody>
</table>
```

©2020 Khronos Group - Rev. 1020
Atomic Functions (continued)

Atomic macros

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

define ATOMIC_FLAG_INIT
Global atomic objects declared with the atomic_flag type can be initialized to a clear state with the ATOMIC_FLAG_INIT macro, for example:

    global atomic_flag guard = ATOMIC_FLAG_INIT;

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 with these extensions enabled: cl_khr_int64_base_atics and cl_khr_int64_extended_aticles

atomic_int Atomic integer types: atomic_int8, atomic_int16, atomic_int32, atomic_int64
atomic_uchar Atomic unsigned integer types: atomic_uchar8, atomic_uchar16, atomic_uchar32, atomic_uchar64
atomic_ushort Atomic unsigned integer types: atomic_ushort8, atomic_ushort16, atomic_ushort32, atomic_ushort64
atomic_ulong Atomic unsigned integer types: atomic_ulong8, atomic_ulong16, atomic_ulong32, atomic_ulong64
atomic_long Atomic signed integer types: atomic_long8, atomic_long16, atomic_long32, atomic_long64
atomic_slong Atomic signed integer types: atomic_slong8, atomic_slong16, atomic_slong32, atomic_slong64
atomic_ulong Atomic unsigned integer types: atomic_uchar8, atomic_uchar16, atomic_uchar32, atomic_uchar64
atomic_ushort Atomic unsigned integer types: atomic_ushort8, atomic_ushort16, atomic_ushort32, atomic_ushort64
atomic_ulong Atomic unsigned integer types: atomic_ulong8, atomic_ulong16, atomic_ulong32, atomic_ulong64
atomic_long Atomic signed integer types: atomic_long8, atomic_long16, atomic_long32, atomic_long64
atomic_slong Atomic signed integer types: atomic_slong8, atomic_slong16, atomic_slong32, atomic_slong64
atomic_int atomic_double Atomic integer and floating-point types: atomic_int8, atomic_int16, atomic_int32, atomic_int64, atomic_double
atomic_int atomic_long Atomic integer and floating-point types: atomic_int8, atomic_int16, atomic_int32, atomic_int64, atomic_long
atomic_int atomic_ulong Atomic integer and floating-point types: atomic_int8, atomic_int16, atomic_int32, atomic_int64, atomic_ulong
atomic_int atomic_uchar Atomic integer and floating-point types: atomic_uchar8, atomic_uchar16, atomic_uchar32, atomic_uchar64
atomic_int atomic_ushort Atomic integer and floating-point types: atomic_uchar8, atomic_uchar16, atomic_uchar32, atomic_uchar64
atomic_int atomic_ulong Atomic integer and floating-point types: atomic_uchar8, atomic_uchar16, atomic_uchar32, atomic_uchar64
atomic_int atomic_uchar Atomic integer and floating-point types: atomic_uchar8, atomic_uchar16, atomic_uchar32, atomic_uchar64
atomic_int atomic_ushort Atomic integer and floating-point types: atomic_uchar8, atomic_uchar16, atomic_uchar32, atomic_uchar64
atomic_int atomic_ulong Atomic integer and floating-point types: atomic_uchar8, atomic_uchar16, atomic_uchar32, atomic_uchar64

Legacy Atomic Functions

These functions provide atomic operations on 32-bit signed and unsigned integers and single precision floating-point to locations in _global or _local memory. T is type int or unsigned int. T may also be type float for atomic_shuffle and, if supported, type long or long for extended 64-bit atomic functions. Q is volatile _global or volatile _local.

atomic_add Q p, T val Read, add, and store
atomic_sub Q p, T val Read, subtract, and store
atomic_xchg Q p, T val Read, swap, and store
atomic_inc Q p Read, increment, and store
atomic_dec Q p Read, decrement, and store
atomic_cmpxchg Q p, T cmp, T val Read, store (store cmp T val ? val : T p)
atomic_min Q p, T val Read, store min(T val, T p)
atomic_max Q p, T val Read, store max(T val, T p)
atomic_and Q p, T val Read, store (T p & T val)
atomic_or Q p, T val Read, store (T p | T val)
atomic_xor Q p, T val Read, store (T p ^ T val)

pipe Built-in Functions

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. Double or vector double types require double precision to be supported. The macro CPX_NULL_RESERVE_ID refers to an invalid reservation ID.

sub_group_all<op>(pipe ptr, T val) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns a non-zero value if predicate evaluates to non-zero for all or any work-items in the work-group.

sub_group_sub<T p, T val> T predicate; where predicate is a vector of boolean values of size T num_packets. Returns true if predicate evaluates to true for all or any work-items in the work-group.

sub_group_scan_inclusive<op>(pipe ptr, T x, size_t size, T val) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the result of reduction operation specified by <op> for all values of x specified by work-items in work-group. <op> may be min, max, or add.

sub_group_reduction<op>(pipe ptr, T p, T x) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the result of reduction operation specified by <op> for all values of x specified by work-items in work-group. <op> may be min, max, or add.

sub_group_something<op>(T x) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the result of reduction operation specified by <op> for all values of x specified by work-items in work-group. <op> may be min, max, or add.

boot is_valid_reserve_id (reserve_id_t reserve_id) bool Returns true if reserve_id is a valid reservation ID and false otherwise.

reserve_id_t reserve_read_pipe (pipe T p, T val) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the result of reduction operation specified by <op> for all values of x specified by work-items in work-group. <op> may be min, max, or add.

reserve_id_t reserve_write_pipe (pipe T p, T val) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the result of reduction operation specified by <op> for all values of x specified by work-items in work-group. <op> may be min, max, or add.

commit_read_pipe (pipe T p, T val) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the result of reduction operation specified by <op> for all values of x specified by work-items in work-group. <op> may be min, max, or add.

commit_write_pipe (pipe T p, T val) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the result of reduction operation specified by <op> for all values of x specified by work-items in work-group. <op> may be min, max, or add.

get_pipe_max_packets (pipe T p) size_t Returns maximum number of packets specified when T p was created.

get_pipe_num_packets (pipe T p) size_t Returns the number of available entries in T p.

Work-group Functions

T is type int, uint, or float. If supported, T can also be type long, ulong, or double. The sub_group_* work-group functions require support for the __opencl_c_work_group_collective_functions feature. All other work-group functions require OpenCL C 2.0 or support for the __opencl_c_work_group_collective_functions feature.

T work_group_broadcast (T a, size_t local_id) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the value of T a to all work-items in the work-group. local_id must be the same value for all work-items in the work-group.

T work_group broadcast (T a, size_t local_id_x, size_t local_id_y, size_t local_id_z) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the value of T a to all work-items in the work-group. local_id must be the same value for all work-items in the work-group.

T work_group broadcast (T a, size_t local_id_x, size_t local_id_y) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the value of T a to all work-items in the work-group. local_id must be the same value for all work-items in the work-group.

T work_group broadcast (T a, size_t local_id_x) T predicate; where predicate is a vector of boolean values of size T num_packets. Returns the value of T a to all work-items in the work-group. local_id must be the same value for all work-items in the work-group.

Notes

---

OpenCL 3.0 Reference Guide

OpenCL C Language

Page 10

©2020 Khronos Group - Rev. 1020

www.khronos.org/opencl
Enqueuing and Kernel Query Built-in Functions [C 6.13.17]
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. The macro CLK_NULL_EVENT refers to an invalid device queue. The macros CLK_NULL_QUEUE refer to an invalid device queue.

The *sub_group* functions require support for the features _opencl_c_subgroups and _opencl_c_device_enqueue. All other functions require support for _opencl_c_device_enqueue or OpenCL C 2.0.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_kernel_work_group_size</td>
<td>Returns the maximum work-group size that can be used to execute a block.</td>
</tr>
<tr>
<td>get_kernel_work_group_size_multiple</td>
<td>Returns the preferred multiple of work-group size for launch.</td>
</tr>
<tr>
<td>create_user_event</td>
<td>Allows a work-item to enqueue a block for execution to queue.</td>
</tr>
<tr>
<td>set_user_event_status</td>
<td>Sets the execution status of a user event.</td>
</tr>
<tr>
<td>void capture_event_profiling_info</td>
<td>Captures profiling information for command associated with event in value.</td>
</tr>
<tr>
<td>enqueue_event_marker</td>
<td>Enqueue a marker command to queue.</td>
</tr>
<tr>
<td>enqueue_kernel</td>
<td>Enqueues work-items to a device.</td>
</tr>
</tbody>
</table>

Helper Built-in Functions [C 6.13.17]
These functions require support for the _opencl_c_device_enqueue feature or OpenCL C 2.0.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void retain_event</td>
<td>Increments event reference count.</td>
</tr>
<tr>
<td>void release_event</td>
<td>Decrements event reference count.</td>
</tr>
<tr>
<td>bool is_valid_event</td>
<td>True for valid event.</td>
</tr>
<tr>
<td>void set_user_event_status</td>
<td>Sets the execution status of a user event.</td>
</tr>
<tr>
<td>uint enqueue_marker</td>
<td>Enqueue a marker command to queue.</td>
</tr>
</tbody>
</table>

Event Built-in Functions [C 6.13.17]
These functions require support for the _opencl_c_device_enqueue feature or OpenCL C 2.0.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int enqueue_kernel</td>
<td>Creates a user event.</td>
</tr>
<tr>
<td>uint get_kernel_work_group_size</td>
<td>Returns the maximum work-group size that can be used to execute a block.</td>
</tr>
<tr>
<td>uint get_kernel_work_group_size_multiple</td>
<td>Returns the preferred multiple of work-group size for launch.</td>
</tr>
<tr>
<td>int enqueue_marker</td>
<td>Creates a user event.</td>
</tr>
</tbody>
</table>

Features and Feature Macros
When an OpenCL optional feature is supported in the language, support will be indicated using a feature.

<table>
<thead>
<tr>
<th>Feature</th>
<th>The OpenCL C compiler supports...</th>
</tr>
</thead>
<tbody>
<tr>
<td>_opencl_c_3d_image_writes</td>
<td>Built-in functions for writing to 3D image objects.</td>
</tr>
<tr>
<td>_opencl_c_atomic_order_acq_rel</td>
<td>Enumerations and built-in functions for atomic operations with acquire and release memory consistency orders.</td>
</tr>
<tr>
<td>_opencl_c_atomic_order_seq_cst</td>
<td>Enumerations and built-in functions for atomic operations and fences with sequentially consistent memory consistency order.</td>
</tr>
<tr>
<td>_opencl_c_atomic_scope_device</td>
<td>Enumerations and built-in functions for atomic operations and fences with device memory scope.</td>
</tr>
<tr>
<td>_opencl_c_atomic_scope_all_svm_devices</td>
<td>Enumerations and built-in functions for atomic operations and fences with all SVM devices memory scope.</td>
</tr>
<tr>
<td>_opencl_c_device_enqueue</td>
<td>Built-in functions to enqueue additional work from the device.</td>
</tr>
<tr>
<td>_opencl_c_fp64</td>
<td>Types and built-in functions with 64-bit floating point types.</td>
</tr>
<tr>
<td>_opencl_c_generic_address_space</td>
<td>The unnamed generic address space.</td>
</tr>
<tr>
<td>_opencl_c_images</td>
<td>Types and built-in functions for images.</td>
</tr>
<tr>
<td>_opencl_c_int64</td>
<td>Types and built-in functions with 64-bit integers.</td>
</tr>
<tr>
<td>_opencl_c_pipes</td>
<td>The pipe modifier and built-in functions to read and write from a pipe.</td>
</tr>
<tr>
<td>_opencl_c_program_scope_global_variables</td>
<td>Program scope variables in the global address space.</td>
</tr>
<tr>
<td>_opencl_c_read_write_images</td>
<td>Reading from and writing to the same image object in a kernel.</td>
</tr>
<tr>
<td>_opencl_c_subgroups</td>
<td>Built-in functions operating on sub-groupings of work-items.</td>
</tr>
<tr>
<td>_opencl_c_work_group_collective_functions</td>
<td>Built-in functions that perform collective operations across a work-group.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Region</th>
<th>Global</th>
<th>Constant</th>
<th>Local</th>
<th>Private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Dynamic allocation</td>
<td>Dynamic allocation</td>
<td>Dynamic allocation</td>
<td>No allocation</td>
</tr>
<tr>
<td></td>
<td>Read/Write access</td>
<td>Read/Write access</td>
<td>No access</td>
<td>No access</td>
</tr>
<tr>
<td>Kernel</td>
<td>No allocation</td>
<td>Static allocation</td>
<td>Static allocation</td>
<td>Static allocation</td>
</tr>
<tr>
<td></td>
<td>Read/Write access</td>
<td>Read-only access</td>
<td>Read/Write access</td>
<td>Read/Write access</td>
</tr>
</tbody>
</table>

©2020 Khronos Group - Rev. 1020
www.khronos.org/opencl

The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices. See www.khronos.org to learn more about the Khronos Group.
OpenCL is a trademark of Apple Inc. and is used under license by Khronos.