OpenCL Overview

The Open Standard for Heterogenous Computing

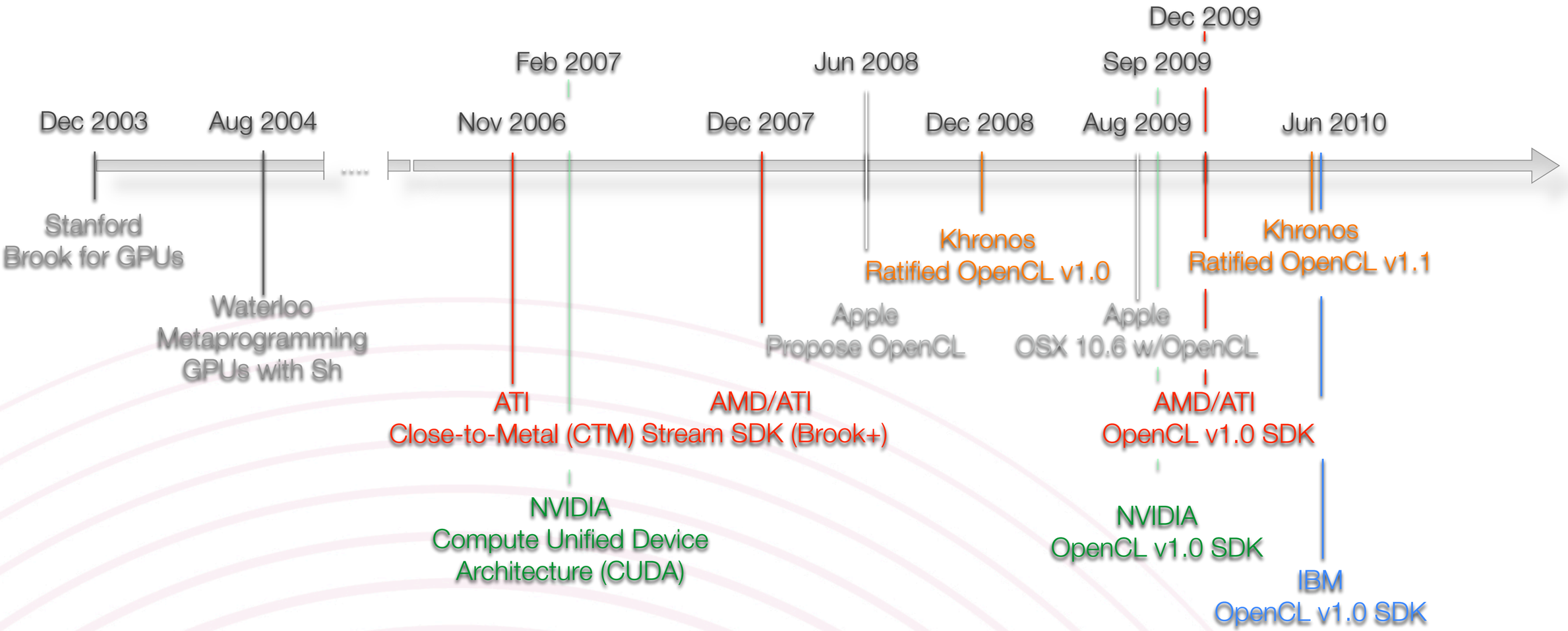
OpenCL:

The Open Computing Language

The open standard for developing cross-platform, vendor agnostic, parallel programs that run on current and future multi-core processors within workstations, desktops, notebooks and embedded devices.

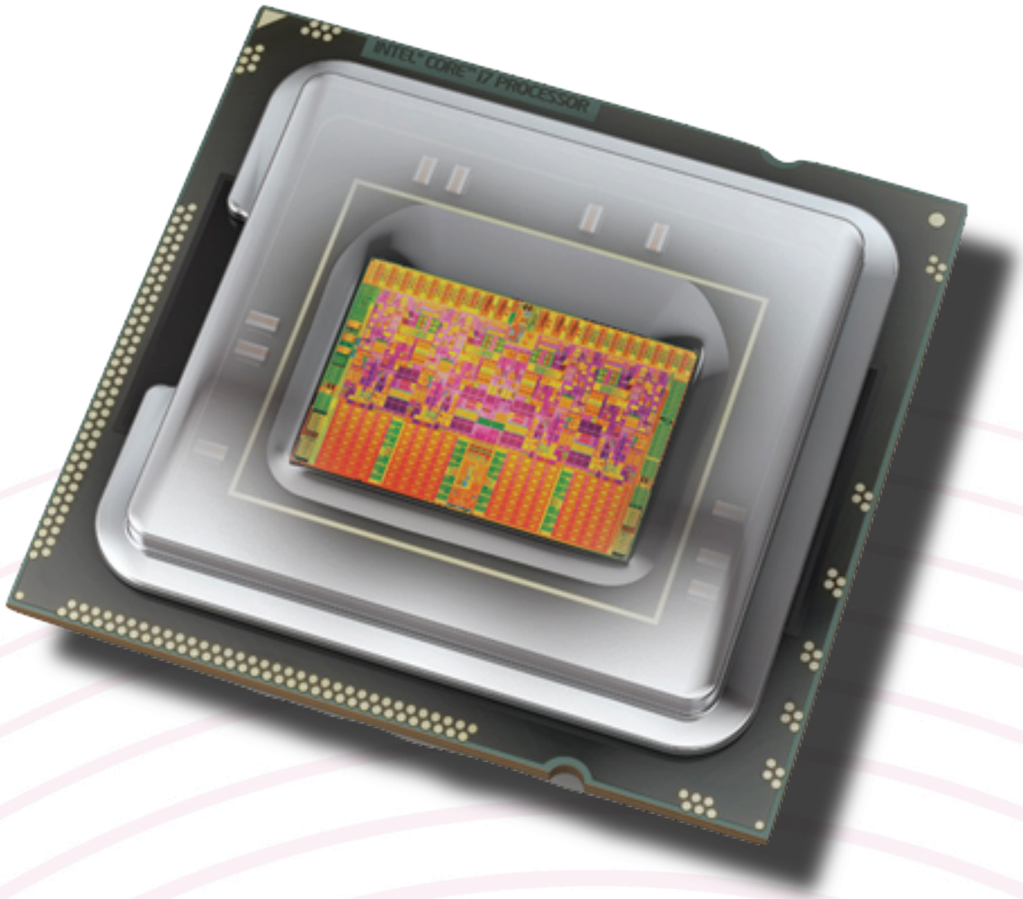
Historical Background

From whence we came and where we're heading...

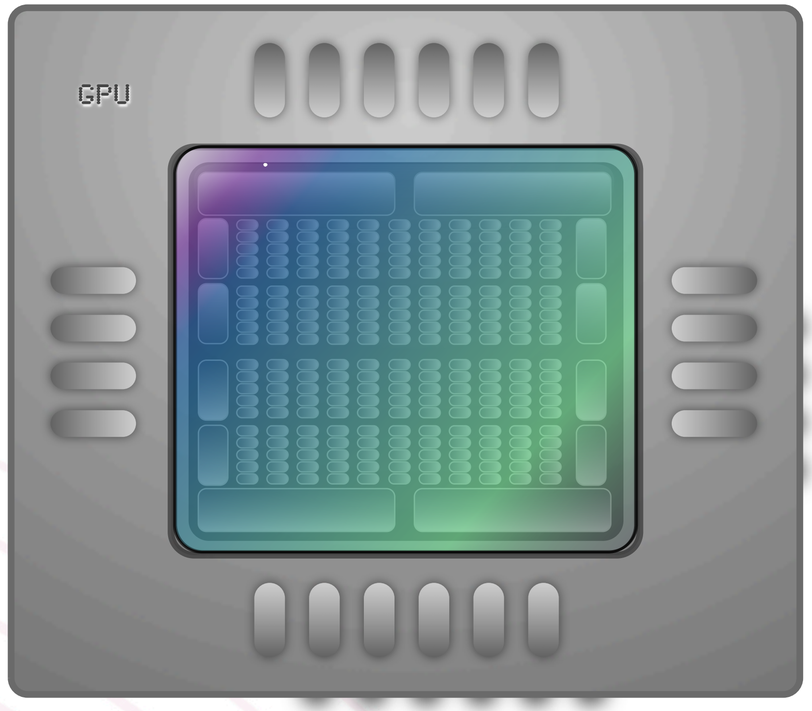


Compute Technology Milestones

Timeline of compute-oriented technology milestones for massively multi-core processors

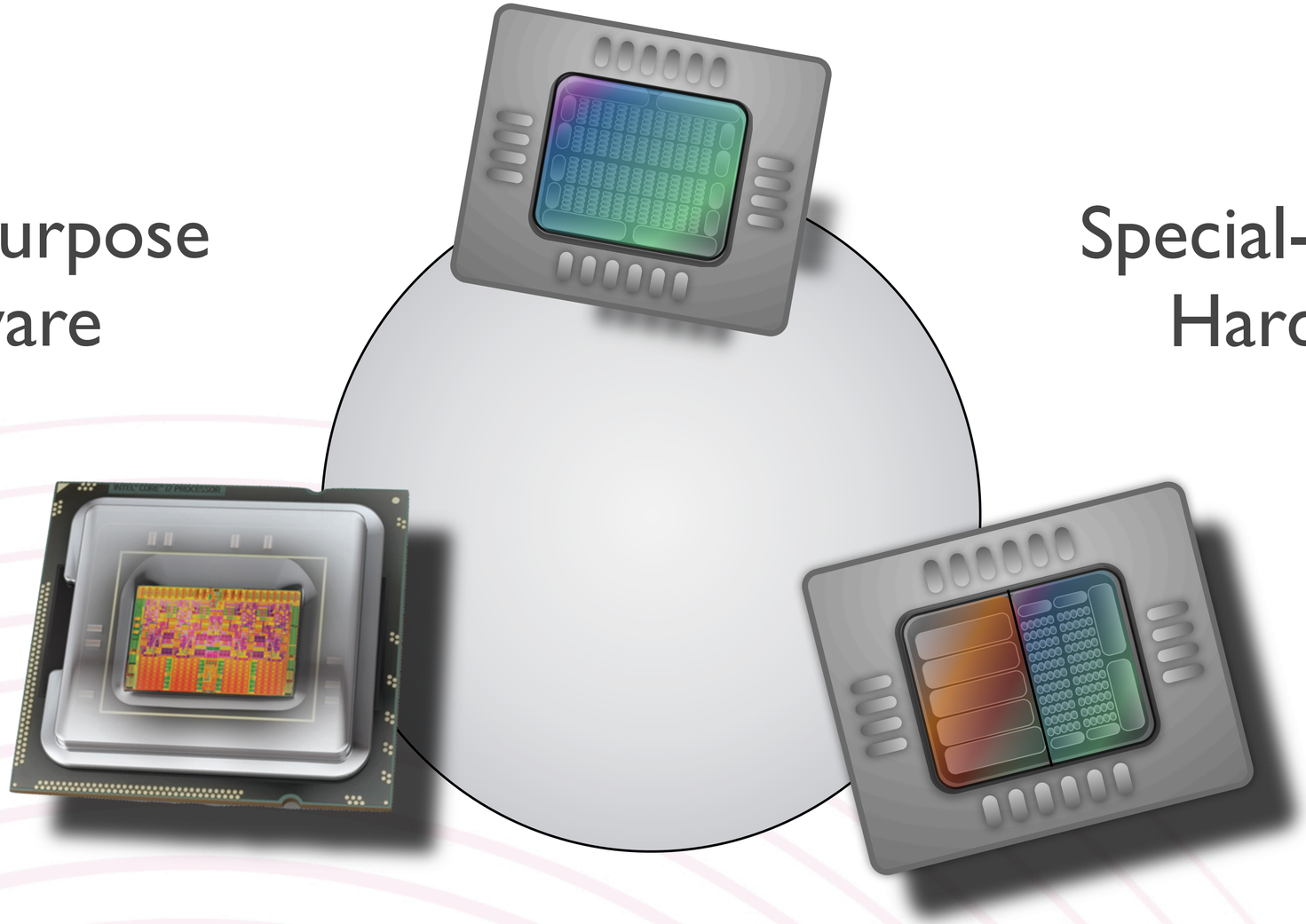


Modern Multi-Core CPUs



Massively Multi-Core GPUs

General-Purpose Hardware



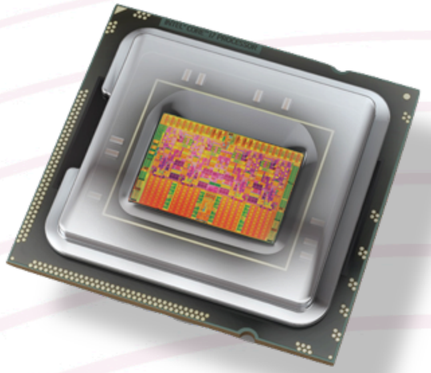
Special-Purpose Hardware

Wheel of Reincarnation

T. H. Myer and I. E. Sutherland. 1968. On the design of display processors. Commun. ACM 11, 6 (June 1968), 410-414.

Application

Multi-Core CPUs



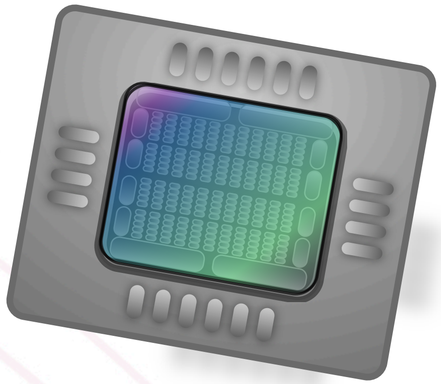
Task-Parallel

Data-Parallel

Serial Code

Graphics

Massively Multi-Core GPUs



Need efficient use of all system resources to enable scalable high-performance applications

Motivation for OpenCL:

Enable heterogenous computing

- Right now on current devices for real application
- Help drive the future for multi-core technology

Design Goals:

Low-level abstraction for compute

- Avoid specifics of the hardware
- Enable high performance
- Support device independence

Design Goals:

Enable all compute resources

- CPUs, GPUs, and other processors
- Data- and task- parallel compute models
- Efficient resource sharing between processors

Design Goals:

Efficient parallel programming model

- ANSI C99 based kernel language
- Minimal restrictions and language additions
- Straight-forward development path

Design Goals:

Consistent results on all platforms

- Well-defined precision requirements for all operations
- Maximal-error allowances for floating-point math
- Comprehensive conformance suite for API + language

Design Goals:

Interoperability with Graphics APIs

- Enable advanced visual computing applications
- Efficient resource sharing for graphics data
- Support graphics oriented built-in methods

Design Goals:

Drive future hardware requirements

- For both consumer-level and high-performance computing applications
- Be prepared for the next spin on the wheel of reincarnation

OpenCL:

Vendor Implementations

AMD: http://developer.amd.com/gpu/ATIStreamSDK/

X86, Radeon, FireStream, FirePro GPUs

APPLE: http://developer.apple.com/object/opencl.html

All shipping desktop + notebook hardware ...

OpenCL:

Vendor Implementations

NVIDIA: http://developer.nvidia.com/object/opencl.html

Tesla, Fermi GPUs

IBM: http://www.alphaworks.ibm.com/tech/opencl

Cell-BE, Power6, Power7, more coming soon...

OpenCL:

Vendor Implementations

INTEL: http://whatif.intel.com/

Intel X86 CPUs

SNU-SAMSUNG: http://opencl.snu.ac.kr/

Cell-BE, ARM, DSP, more coming soon...

Anatomy of OpenCL:

Technology stack

- Language specification
- Platform specification
- Runtime specification

Khronos.org:

OpenCL Quick Reference Card

http://www.khronos.org/files/opencl-1-1-quick-reference-card.pdf

OpenCL API 1.1 Quick Reference Card - Page 1

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

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencl.

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue **clCreateCommandQueue** (
cl_context *context*, cl_device_id *device*,
cl_command_queue_properties *properties*,
cl_int *errcode_ret*)

properties: CL_QUEUE_PROFILING_ENABLE,
CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

cl_int **clRetainCommandQueue** (
cl_command_queue *command_queue*)

cl_int **clReleaseCommandQueue** (
cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (
cl_command_queue *command_queue*,
cl_command_queue_info *param_name*,
size_t *param_value_size*, void *param_value,
size_t *param_value_size_ret*)

param_name: CL_QUEUE_CONTEXT,
CL_QUEUE_DEVICE,
CL_QUEUE_REFERENCE_COUNT,
CL_QUEUE_PROPERTIES

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.

Contexts [4.3]

cl_context **clCreateContext** (
const cl_context_properties *properties, cl_uint *num_devices*,
const cl_device_id *devices, void (CL_CALLBACK*pfn_notify)
(const char *errinfo, const void *private_info,
size_t *cb*, void *user_data*),
void *user_data, cl_int *errcode_ret*)

properties: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR,
CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR,
CL_WGL_HDC_KHR

cl_context **clCreateContextFromType** (
const cl_context_properties *properties,
cl_device_type *device_type*, void (CL_CALLBACK *pfn_notify)
(const char *errinfo, const void *private_info*, size_t *cb*,
void *user_data*),
void *user_data*, cl_int *errcode_ret*)

properties: See **clCreateContext**

cl_int **clRetainContext** (cl_context *context*)

cl_int **clReleaseContext** (cl_context *context*)

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*)

param_name: CL_CONTEXT_REFERENCE_COUNT,
CL_CONTEXT_{DEVICES, PROPERTIES}, CL_CONTEXT_NUM_DEVICES

Querying Platform Info and Devices [4.1, 4.2]

cl_int **clGetPlatformIDs** (cl_uint *num_entries*,
cl_platform_id *platforms*, cl_uint *num_platforms*)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*,
cl_platform_info *param_name*, size_t *param_value_size*,
void *param_value*, size_t *param_value_size_ret*)

param_name: CL_PLATFORM_{PROFILE, VERSION},
CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_int **clGetDeviceIDs** (cl_platform_id *platform*,
cl_device_type *device_type*, cl_uint *num_entries*,
cl_device_id *devices*, cl_uint *num_devices*)

device_type: CL_DEVICE_TYPE_{CPU, GPU},
CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

cl_int **clGetDeviceInfo** (cl_device_id *device*,
cl_device_info *param_name*, size_t *param_value_size*,
void *param_value*, size_t *param_value_size_ret*)

param_name: CL_DEVICE_TYPE,
CL_DEVICE_VENDOR_ID,
CL_DEVICE_MAX_COMPUTE_UNITS,
CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES},
CL_DEVICE_MAX_WORK_GROUP_SIZE,
CL_DEVICE_{NATIVE_PREFERRED}_VECTOR_WIDTH_CHAR,
CL_DEVICE_{NATIVE_PREFERRED}_VECTOR_WIDTH_SHORT,
CL_DEVICE_{NATIVE_PREFERRED}_VECTOR_WIDTH_INT,
CL_DEVICE_{NATIVE_PREFERRED}_VECTOR_WIDTH_LONG,
CL_DEVICE_{NATIVE_PREFERRED}_VECTOR_WIDTH_FLOAT,
CL_DEVICE_{NATIVE_PREFERRED}_VECTOR_WIDTH_DOUBLE,
CL_DEVICE_{NATIVE_PREFERRED}_VECTOR_WIDTH_HALF,
CL_DEVICE_MAX_CLOCK_FREQUENCY,
CL_DEVICE_ADDRESS_BITS,
CL_DEVICE_MAX_MEM_ALLOC_SIZE,
CL_DEVICE_IMAGE_SUPPORT,
CL_DEVICE_MAX_{READ, WRITE}_IMAGE_ARGS,
CL_DEVICE_IMAGE2D_MAX_{WIDTH, HEIGHT},
CL_DEVICE_IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH},
CL_DEVICE_MAX_SAMPLERS,
CL_DEVICE_MAX_PARAMETER_SIZE,
CL_DEVICE_MEM_BASE_ADDR_ALIGN,
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE,
CL_DEVICE_SINGLE_FP_CONFIG,
CL_DEVICE_GLOBAL_MEM_CACHE_{TYPE, SIZE},
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE,
CL_DEVICE_GLOBAL_MEM_SIZE,
CL_DEVICE_MAX_CONSTANT_{BUFFER_SIZE, ARGS}
CL_DEVICE_LOCAL_MEM_{TYPE, SIZE},
CL_DEVICE_ERROR_CORRECTION_SUPPORT,
CL_DEVICE_PROFILING_TIMER_RESOLUTION,
CL_DEVICE_ENDIAN_LITTLE,
CL_DEVICE_AVAILABLE,
CL_DEVICE_COMPILER_AVAILABLE,
CL_DEVICE_EXECUTION_CAPABILITIES,
CL_DEVICE_QUEUE_PROPERTIES,
CL_DEVICE_{NAME, VENDOR, PROFILE, EXTENSIONS},
CL_DEVICE_HOST_UNIFIED_MEMORY,
CL_DEVICE_OPENCL_C_VERSION,
CL_DEVICE_VERSION,
CL_DRIVER_VERSION, CL_DEVICE_PLATFORM

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem **clCreateBuffer** (cl_context *context*,
cl_mem_flags *flags*, size_t *size*, void *host_ptr*,
cl_int *errcode_ret*)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*,
cl_mem_flags *flags*,
cl_buffer_create_type *buffer_create_type*,
const void *buffer_create_info*, cl_int *errcode_ret*)

flags for **clCreateBuffer** and **clCreateSubBuffer**:
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int **clEnqueueReadBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, size_t *offset*, size_t *cb*,
void *ptr*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*, cl_event *event*)

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

cl_int **clEnqueueReadBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, const size_t *buffer_origin*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void *ptr, cl_uint *num_events_in_wait_list*,
const 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*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void *ptr, cl_uint *num_events_in_wait_list*,
const 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 *cb*,
cl_uint *num_events_in_wait_list*,
const 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*[3], const size_t *dst_origin*[3],
const size_t *region*[3], size_t *src_row_pitch*,
size_t *src_slice_pitch*, size_t *dst_row_pitch*,
size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*, cl_event *event*)

Map Buffer Objects [5.2.2]

void * **clEnqueueMapBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_map*, cl_map_flags *map_flags*,
size_t *offset*, size_t *cb*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list*, cl_event *event*,
cl_int *errcode_ret*)

Map Buffer Objects [5.4.1-2]

cl_int **clRetainMemObject** (cl_mem *memobj*)

cl_int **clReleaseMemObject** (cl_mem *memobj*)

cl_int **clSetMemObjectDestructorCallback** (
cl_mem *memobj*, void (CL_CALLBACK *pfn_notify)
(cl_mem *memobj*, void *user_data),
void *user_data*)

cl_int **clEnqueueUnmapMemObject** (
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*)

Query Buffer Object [5.4.3]

cl_int **clGetMemObjectInfo** (cl_mem *memobj*,
cl_mem_info *param_name*, size_t *param_value_size*,
void *param_value*, size_t *param_value_size_ret*)

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
CL_MEM_{MAP_REFERENCE}_COUNT, CL_MEM_OFFSET,
CL_MEM_CONTEXT, CL_MEM_ASSOCIATED_MEMOBJECT

Program Objects

Create Program Objects [5.6.1]

cl_program **clCreateProgramWithSource** (
cl_context *context*, cl_uint *count*, const char **strings*,
const size_t *lengths*, cl_int *errcode_ret*)

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

cl_int **clRetainProgram** (cl_program *program*)

cl_int **clReleaseProgram** (cl_program *program*)

Build Program Executable [5.6.2]

cl_int **clBuildProgram** (cl_program *program*,
cl_uint *num_devices*, const cl_device_id *device_list*,
const char *options*, void (CL_CALLBACK*pfn_notify)
(cl_program *program*, void *user_data*),
void *user_data*)

Build Options [5.6.3]

Preprocessor: *(-D processed in order listed in clBuildProgram)*
-D name -D name=definition -I dir

Optimization options:
-cl-opt-disable
-cl-strict-aliasing
-cl-mad-enable
-cl-no-signed-zeros
-cl-finite-math-only
-cl-fast-relaxed-math
-cl-unsafe-math-optimizations

Math Intrinsics:
-cl-single-precision-constant
-cl-denorms-are-zero

Warning request/suppress:
-w
-Werror

Control OpenCL C language version:
-cl-std=CL1.1 // OpenCL 1.1 specification.

Query Program Objects [5.6.5]

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*)

param_name: CL_PROGRAM_{REFERENCE_COUNT},
CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES},
CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES}

(Program Objects Continue >)

OpenCL API 1.1 Quick Reference Card - Page 1

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

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencl.

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue **clCreateCommandQueue** (
  cl_context *context*, cl_device_id *device*,
  cl_command_queue_properties *properties*,
  cl_int *errcode_ret*)

properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

cl_int **clRetainCommandQueue** (
  cl_command_queue *command_queue*)

cl_int **clReleaseCommandQueue** (
  cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (
  cl_command_queue *command_queue*,
  cl_command_queue_info *param_name*,
  size_t *param_value_size*, void *param_value*,
  size_t *param_value_size_ret*)

param_name: CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE, CL_QUEUE_REFERENCE_COUNT, CL_QUEUE_PROPERTIES

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.

Contexts [4.3]

cl_context **clCreateContext** (
  const cl_context_properties *properties, cl_uint *num_devices*,
  const cl_device_id *devices, void (CL_CALLBACK*pfn_notify)
    (const char *errinfo, const void *private_info,
    size_t *cb*, void *user_data*),
  void *user_data*, cl_int *errcode_ret*)

properties: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR, CL_WGL_HDC_KHR

cl_context **clCreateContextFromType** (
  const cl_context_properties *properties,
  cl_device_type *device_type*, void (CL_CALLBACK *pfn_notify)
    (const char *errinfo, const void *private_info, size_t *cb*,
    void *user_data*),
  void *user_data*, cl_int *errcode_ret*)

properties: See **clCreateContext**

cl_int **clRetainContext** (cl_context *context*)

cl_int **clReleaseContext** (cl_context *context*)

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*)

param_name: CL_CONTEXT_REFERENCE_COUNT, CL_CONTEXT_{DEVICES, PROPERTIES}, CL_CONTEXT_NUM_DEVICES

Querying Platform Info and Devices [4.1, 4.2]

cl_int **clGetPlatformIDs** (cl_uint *num_entries*,
  cl_platform_id *platforms*, cl_uint *num_platforms*)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*,
  cl_platform_info *param_name*, size_t *param_value_size*,
  void *param_value*, size_t *param_value_size_ret*)

param_name: CL_PLATFORM_{PROFILE, VERSION}, CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_int **clGetDeviceIDs** (cl_platform_id *platform*,
  cl_device_type *device_type*, cl_uint *num_entries*,
  cl_device_id *devices, cl_uint *num_devices*)

device_type: CL_DEVICE_TYPE_{CPU, GPU}, CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

cl_int **clGetDeviceInfo** (cl_device_id *device*,
  cl_device_info *param_name*, size_t *param_value_size*,
  void *param_value*, size_t *param_value_size_ret*)

param_name: CL_DEVICE_TYPE, CL_DEVICE_VENDOR_ID, CL_DEVICE_MAX_COMPUTE_UNITS, CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES}, CL_DEVICE_MAX_WORK_GROUP_SIZE, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_CHAR, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_SHORT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_INT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_LONG, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_FLOAT, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_DOUBLE, CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_HALF, CL_DEVICE_MAX_CLOCK_FREQUENCY, CL_DEVICE_ADDRESS_BITS, CL_DEVICE_MAX_MEM_ALLOC_SIZE, CL_DEVICE_IMAGE_SUPPORT, CL_DEVICE_MAX_{READ, WRITE}_IMAGE_ARGS, CL_DEVICE_{IMAGE2D_MAX_{WIDTH, HEIGHT}, CL_DEVICE_IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH}, CL_DEVICE_MAX_SAMPLERS, CL_DEVICE_MAX_PARAMETER_SIZE, CL_DEVICE_MEM_BASE_ADDR_ALIGN, CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE, CL_DEVICE_SINGLE_FP_CONFIG, CL_DEVICE_GLOBAL_MEM_CACHE_{TYPE, SIZE}, CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE, CL_DEVICE_GLOBAL_MEM_SIZE, CL_DEVICE_MAX_CONSTANT_{BUFFER_SIZE, ARGS} CL_DEVICE_LOCAL_MEM_{TYPE, SIZE}, CL_DEVICE_ERROR_CORRECTION_SUPPORT, CL_DEVICE_PROFILING_TIMER_RESOLUTION, CL_DEVICE_ENDIAN_LITTLE, CL_DEVICE_AVAILABLE, CL_DEVICE_COMPILER_AVAILABLE, CL_DEVICE_EXECUTION_CAPABILITIES, CL_DEVICE_QUEUE_PROPERTIES, CL_DEVICE_{NAME, VENDOR, PROFILE, EXTENSIONS}, CL_DEVICE_HOST_UNIFIED_MEMORY, CL_DEVICE_OPENCL_C_VERSION, CL_DEVICE_VERSION, CL_DRIVER_VERSION, CL_DEVICE_PLATFORM

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem **clCreateBuffer** (cl_context *context*,
  cl_mem_flags *flags*, size_t *size*, void *host_ptr*,
  cl_int *errcode_ret*)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*,
  cl_mem_flags *flags*,
  cl_buffer_create_type *buffer_create_type*,
  const void *buffer_create_info*, cl_int *errcode_ret*)

flags for **clCreateBuffer** and **clCreateSubBuffer**: CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int **clEnqueueReadBuffer** (
  cl_command_queue *command_queue*, cl_mem *buffer*,
  cl_bool *blocking_read*, size_t *offset*, size_t *cb*,
  void *ptr*, cl_uint *num_events_in_wait_list*,
  const cl_event *event_wait_list*, cl_event *event*)

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

cl_int **clEnqueueReadBufferRect** (
  cl_command_queue *command_queue*, cl_mem *buffer*,
  cl_bool *blocking_read*, const size_t *buffer_origin*[3],
  const size_t *host_origin*[3], const size_t *region*[3],
  size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
  size_t *host_row_pitch*, size_t *host_slice_pitch*,
  void *ptr, cl_uint *num_events_in_wait_list*,
  const 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*[3],
  const size_t *host_origin*[3], const size_t *region*[3],
  size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
  size_t *host_row_pitch*, size_t *host_slice_pitch*,
  void *ptr, cl_uint *num_events_in_wait_list*,
  const 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 *cb*,
  cl_uint *num_events_in_wait_list*,
  const 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*[3], const size_t *dst_origin*[3],
  const size_t *region*[3], size_t *src_row_pitch*,
  size_t *src_slice_pitch*, size_t *dst_row_pitch*,
  size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*,
  const cl_event *event_wait_list*, cl_event *event*)

Map Buffer Objects [5.2.2]

void * **clEnqueueMapBuffer** (
  cl_command_queue *command_queue*, cl_mem *buffer*,
  cl_bool *blocking_map*, cl_map_flags *map_flags*,
  size_t *offset*, size_t *cb*, cl_uint *num_events_in_wait_list*,
  const cl_event *event_wait_list*, cl_event *event*,
  cl_int *errcode_ret*)

Map Buffer Objects [5.4.1-2]

cl_int **clRetainMemObject** (cl_mem *memobj*)

cl_int **clReleaseMemObject** (cl_mem *memobj*)

cl_int **clSetMemObjectDestructorCallback** (
  cl_mem *memobj*, void (CL_CALLBACK *pfn_notify)
    (cl_mem *memobj*, void *user_data),
  void *user_data*)

cl_int **clEnqueueUnmapMemObject** (
  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*)

Query Buffer Object [5.4.3]

cl_int **clGetMemObjectInfo** (cl_mem *memobj*,
  cl_mem_info *param_name*, size_t *param_value_size*,
  void *param_value*, size_t *param_value_size_ret*)

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR}, CL_MEM_{MAP_REFERENCE}_COUNT, CL_MEM_OFFSET, CL_MEM_CONTEXT, CL_MEM_ASSOCIATED_MEMOBJECT

Program Objects

Create Program Objects [5.6.1]

cl_program **clCreateProgramWithSource** (
  cl_context *context*, cl_uint *count*, const char **strings*,
  const size_t *lengths*, cl_int *errcode_ret*)

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

cl_int **clRetainProgram** (cl_program *program*)

cl_int **clReleaseProgram** (cl_program *program*)

Build Program Executable [5.6.2]

cl_int **clBuildProgram** (cl_program *program*,
  cl_uint *num_devices*, const cl_device_id *device_list*,
  const char *options*, void (CL_CALLBACK*pfn_notify)
    (cl_program *program*, void *user_data*),
  void *user_data*)

Build Options [5.6.3]

Preprocessor: *(-D processed in order listed in clBuildProgram)*
-D name -D name=definition -I dir

Optimization options:
-cl-opt-disable
-cl-strict-aliasing
-cl-mad-enable
-cl-no-signed-zeros
-cl-finite-math-only
-cl-fast-relaxed-math
-cl-unsafe-math-optimizations

Math Intrinsics:
-cl-single-precision-constant
-cl-denorms-are-zero

Warning request/suppress:
-w
-Werror

Control OpenCL C language version:
-cl-std=CL1.1 // OpenCL 1.1 specification.

Query Program Objects [5.6.5]

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*)

param_name: CL_PROGRAM_{REFERENCE_COUNT}, CL_PROGRAM_{CONTEXT, NUM_DEVICES, DEVICES}, CL_PROGRAM_{SOURCE, BINARY_SIZES, BINARIES}

(Program Objects Continue >)

OpenCL API 1.1 Quick Reference Card - Page 2

Program Objects (continued)

cl_int **clGetProgramBuildInfo** (cl_program *program*,
  cl_device_id *device*, cl_program_build_info *param_name*,
  size_t *param_value_size*, void *param_value*,
  size_t *param_value_size_ret*)

param_name: CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG}

Unload the OpenCL Compiler [5.6.4]

cl_int **clUnloadCompiler** (void)

Supported Data Types

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_half	16-bit float (for storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	void	

Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
charn	cl_charn	8-bit signed
ucharn	cl_ucharn	8-bit unsigned
shortn	cl_shortn	16-bit signed
ushortn	cl_ushortn	16-bit unsigned
intn	cl_intn	32-bit signed
uintn	cl_uintn	32-bit unsigned
longn	cl_longn	64-bit signed
ulongn	cl_ulongn	64-bit unsigned
floatn	cl_floatn	32-bit float

Other Built-in Data Types [6.1.3]

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
sampler_t	sampler handle
event_t	event handle

Reserved Data Types [6.1.4]

OpenCL Type	Description
booln	boolean vector
double, doublen OPTIONAL	64-bit float, vector
halfn	16-bit, vector
quad, quadn	128-bit float, vector
complex half, complex halfn imaginary half, imaginary halfn	16-bit complex, vector
complex float, complex floatn imaginary float, imaginary floatn	32-bit complex, vector
complex double, complex doublen imaginary double, imaginary doublen	64-bit complex, vector
complex quad, complex quadn imaginary quad, imaginary quadn	128-bit complex, vector
floatnxm	n*m matrix of 32-bit floats
doublenxm	n*m matrix of 64-bit floats
long double, long doublen	64 - 128-bit float, vector
long long, long longnb	128-bit signed
unsigned long long, ulong long, ulong longn	128-bit unsigned

Kernel and Event Objects

Create Kernel Objects [5.7.1]

cl_kernel **clCreateKernel** (cl_program *program*,
  const char *kernel_name*, cl_int *errcode_ret*)

cl_int **clCreateKernelsInProgram** (cl_program *program*,
  cl_uint *num_kernels*, cl_kernel *kernels*,
  cl_uint *num_kernels_ret*)

cl_int **clRetainKernel** (cl_kernel *kernel*)

cl_int **clReleaseKernel** (cl_kernel *kernel*)

Kernel Args. & Object Queries [5.7.2, 5.7.3]

cl_int **clSetKernelArg** (cl_kernel *kernel*, cl_uint *arg_index*,
  size_t *arg_size*, const void *arg_value*)

cl_int **clGetKernelInfo** (cl_kernel *kernel*,
  cl_kernel_info *param_name*, size_t *param_value_size*,
  void *param_value*, size_t *param_value_size_ret*)

param_name: CL_KERNEL_FUNCTION_NAME, CL_KERNEL_NUM_ARGS, CL_KERNEL_REFERENCE_COUNT, CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM

cl_int **clGetKernelWorkGroupInfo** (
  cl_kernel *kernel*, cl_device_id *device*,
  cl_kernel_work_group_info *param_name*,
  size_t *param_value_size*, void *param_value*,
  size_t *param_value_size_ret*)

param_name: CL_KERNEL_WORK_GROUP_SIZE, CL_KERNEL_COMPILE_WORK_GROUP_SIZE, CL_KERNEL_{LOCAL, PRIVATE}_MEM_SIZE, CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE

Execute Kernels [5.8]

cl_int **clEnqueueNDRangeKernel** (
  cl_command_queue *command_queue*,
  cl_kernel *kernel*, cl_uint *work_dim*,
  const size_t *global_work_offset*,
  const size_t *global_work_size*,
  const size_t *local_work_size*,
  cl_uint *num_events_in_wait_list*,
  const cl_event *event_wait_list*, cl_event *event*)

cl_int **clEnqueueTask** (
  cl_command_queue *command_queue*, cl_kernel
  kernel, cl_uint *num_events_in_wait_list*,
  const cl_event *event_wait_list*, cl_event *event*)

cl_int **clEnqueueNativeKernel** (cl_command_queue
  command_queue, void (**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*)

Event Objects [5.9]

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 *param_name*, size_t *param_value_size*,
  void *param_value*, size_t *param_value_size_ret*)

param_name: CL_EVENT_COMMAND_{QUEUE, TYPE}, CL_EVENT_{CONTEXT, REFERENCE_COUNT}, CL_EVENT_COMMAND_EXECUTION_STATUS

cl_int **clSetEventCallback** (cl_event *event*,
  cl_int *command_exec_callback_type*,
  void (CL_CALLBACK *pfn_event_notify)
    (cl_event *event*, cl_int *event_command_exec_status*,
    void *user_data*),
  void *user_data*)

cl_int **clRetainEvent** (cl_event *event*)

cl_int **clReleaseEvent** (cl_event *event*)

Out-of-order Execution of Kernels & Memory Object Commands [5.10]

cl_int **clEnqueueMarker** (
  cl_command_queue *command_queue*,
  cl_event *event*)

cl_int **clEnqueueWaitForEvents** (
  cl_command_queue *command_queue*,
  cl_uint *num_events*, const cl_event *event_list*)

cl_int **clEnqueueBarrier** (
  cl_command_queue *command_queue*)

Profiling Operations [5.11]

cl_int **clGetEventProfilingInfo** (cl_event *event*,
  cl_profiling_info *param_name*,
  size_t *param_value_size*, void *param_value*,
  size_t *param_value_size_ret*)

param_name: CL_PROFILING_COMMAND_QUEUED, CL_PROFILING_COMMAND_{SUBMIT, START, END}

Flush and Finish [5.12]

cl_int **clFlush** (cl_command_queue *command_queue*)

cl_int **clFinish** (cl_command_queue *command_queue*)

Vector Component Addressing [6.1.7]

Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	v.x, v.s0	v.y, v.s1														
float3 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2													
float4 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2	v.w, v.s3												
float8 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7								
float16 v;	v.s0	v.s1	v.s2	v.s3	v.s4	v.s5	v.s6	v.s7	v.s8	v.s9	v.sa, v.sA	v.sb, v.sB	v.sc, v.sC	v.sd, v.sD	v.se, v.sE	v.sf, v.sF

Vector Addressing Equivalencies

Numeric indices are preceded by the letter s or S, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: v.yx, v.xx, v.lo.x

	v.lo	v.hi	v.odd	v.even
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x, v.s0
float3 *	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s02, v.xz

	v.lo	v.hi	v.odd	v.even
float8	v.s0123	v.s4567	v.s1357	v.s0246
float16	v.s01234567	v.s89abcdef	v.s13579bdf	v.s02468ace

* When using .lo or .hi with a 3-component vector, the .w component is undefined.

Conversions & Type Casting Examples [6.2]

T a = convert_*T*(*b*); *// Scalar to scalar, or scalar to vector*
T a = convert_*T*(*b*);
T a = convert_*T*_*R*(*b*);
T a = as_*T*(*b*);

T a = convert_*T*_sat_*R*(*b*); *//R is rounding mode*

R can be one of the following rounding modes:

_rte	to nearest even	_rtp	toward + infinity
_rtz	toward zero	_rtn	toward - infinity

Operators [6.3]

These operators behave similarly as in C99 except that operands may include vector types when possible:

+	-	*	%	/	--	++	==	!=	&
~	^	>	<	>=	<=	|	!	&&	||
?:	>>	<<	,	=	*op*=	*sizeof*			

Address Space Qualifiers [6.5]

__global, global __local, local
__constant, constant __private, private

Function Qualifiers [6.7]

__kernel, kernel
__attribute__((vec_type_hint(*type*))) *//type defaults to int*
__attribute__((work_group_size_hint(*X, Y, Z*)))
__attribute__((reqd_work_group_size(*X, Y, Z*)))

Reference card production by Miller & Mattson www.millermattson.com

Scalar Storage Types

Data Types for the OpenCL Framework

```
void       - incomplete type corresponding to empty set

cl_char    - 8bit signed two's complement integer value
cl_uchar   - 8bit unsigned integer

cl_short   - 16bit signed two's complement integer value
cl_ushort  - 16bit unsigned integer

cl_int     - 32bit signed two's complement integer value
cl_uint    - 32bit unsigned integer

cl_long    - 64bit signed two's complement integer value
cl_ulong   - 64bit unsigned integer value

cl_half    - half precision floating-point value (IEEE 754-2008)
cl_float   - full precision floating-point value (IEEE 754)
cl_double  - double precision floating-point value (IEEE 754)
```

Vector Storage Types

Data Types for the OpenCL Framework

N – Supported values of N are 2,4,8,16.

`cl_char`***N*** – 8bit signed two's complement integer value
`cl_uchar`***N*** – 8bit unsigned integer

`cl_short`***N*** – 16bit signed two's complement integer value
`cl_ushort`***N*** – 16bit unsigned integer

`cl_int`***N*** – 32bit signed two's complement integer value
`cl_uint`***N*** – 32bit unsigned integer

`cl_long`***N*** – 64bit signed two's complement integer value
`cl_ulong`***N*** – 64bit unsigned integer value

`cl_half`***N*** – half precision floating-point value (**IEEE 754-2008**)
`cl_float`***N*** – full precision floating-point value (**IEEE 754**)
`cl_double`***N*** – double precision floating-point value (**IEEE 754**)

Object Types

Data Types for the OpenCL Framework

```
cl_platform_id    - identifier for a specific platform
cl_device_id      - identifier for a specific compute device
cl_context        - handle for a compute context
cl_command_queue  - handle for a command queue (for compute device)
cl_mem            - handle for a memory resource (managed by the context)
cl_program        - handle for a program resource (library of kernels)
cl_kernel         - handle for a compute kernel (compiled
```

All object types are opaque handles

Enables cross-platform compatibility for complex data types

All objects are reference counted and garbage collected

When reference count reaches zero, object is deallocated

Enqueue Command Methods

Conventions for the OpenCL Framework

```
cl_int clEnqueueMethod(cl_command_queue, /* command queue */,
                       ...                /* method specific parameters */,
                       cl_uint *          /* num of events in wait list */,
                       const cl_event *   /* event wait list */,
                       cl_event *         /* returned event */)
```

All enqueue methods return an error code

Method returns CL_SUCCESS if command was enqueued successfully

All enqueue methods support an event wait-list

Command will not get executed until all events in list are complete

All enqueue methods return an event for the command

Returned event identifies the specific command that was enqueued

Create Object Methods

Conventions for the OpenCL Framework

```
cl_object clCreateMethod(cl_context /* compute context */,
                         ...        /* method specific parameters */,
                         cl_int *   /* returned error code */)
```

All object creation methods return an object handle

Method returns an invalid object if creation fails

All object creation methods require a compute context

Objects and resources are managed in a context

All object creation methods optionally return an error code

Returned value identifies any errors that were encountered

Value is set to CL_SUCCESS if object was created successfully

Object Management Methods

Conventions for the OpenCL Framework

```
cl_int clRetainObject( cl_object      /* handle of object to retain */ )
```

```
cl_int clReleaseObject( cl_object     /* handle of object to release */ )
```

All retain / release object methods return an error code

Method returns CL_SUCCESS if operation was successfully

All retain methods increment the object reference count

Reference count determines life span and is set to one upon creation of object

All release methods decrement the object reference count

When reference count reaches zero, object is deallocated

Kernel Language

ANSI C99-based language

- Familiar to developers
- Some extensions & restrictions
- Includes a rich set of built-in functions
- Supports online & offline compilation

```
__kernel void square(
    __global float* input, __global float* output)
{
    size_t i = get_global_id(0);
    output[i] = input[i] * input[i];
}
```

Declare functions with __kernel attribute

Defines an entry point or exported method in a program object

Use address space and usage qualifiers for memory

Address spaces and data usage must be specified for all memory objects

Built-in methods provide access to index within compute domain

Use **get_global_id** for unique work-item id, **get_group_id** for work-group, etc

Kernel Language

Well-defined precision requirements

- All math functions have accuracy requirements
- IEEE 754 floating-point rounding w/limits on max error
- Support for half and double precision

OpenCL Platform

Hardware abstraction layer

- Access a diverse set of compute resources
- Device agnostic and vendor neutral
- Cross-platform and driver independent

OpenCL Runtime

Management interface for resources

- Query, select and setup devices
- Create and manage compute contexts
- Allocate, read and write memory objects
- Execute work and synchronise devices

```
// Fill our data set with random float values
int count = 1024 * 1024;
for(i = 0; i < count; i++)
    data[i] = rand() / (float)RAND_MAX;

// Connect to a compute device, create a context and a command queue
cl_device_id device;
clGetDeviceIDs(CL_DEVICE_TYPE_GPU, 1, &device, NULL);
cl_context context = clCreateContext(0, 1, & device, NULL, NULL, NULL);
cl_command_queue queue = clCreateCommandQueue(context, device, 0, NULL);

// Create and build a program from our OpenCL-C source code
cl_program program = clCreateProgramWithSource(context, 1, (const char **) &src,
                                                NULL, NULL);
clBuildProgram(program, 0, NULL, NULL, NULL, NULL);

// Create a kernel from our program
cl_kernel kernel = clCreateKernel(program, "square", NULL);
```

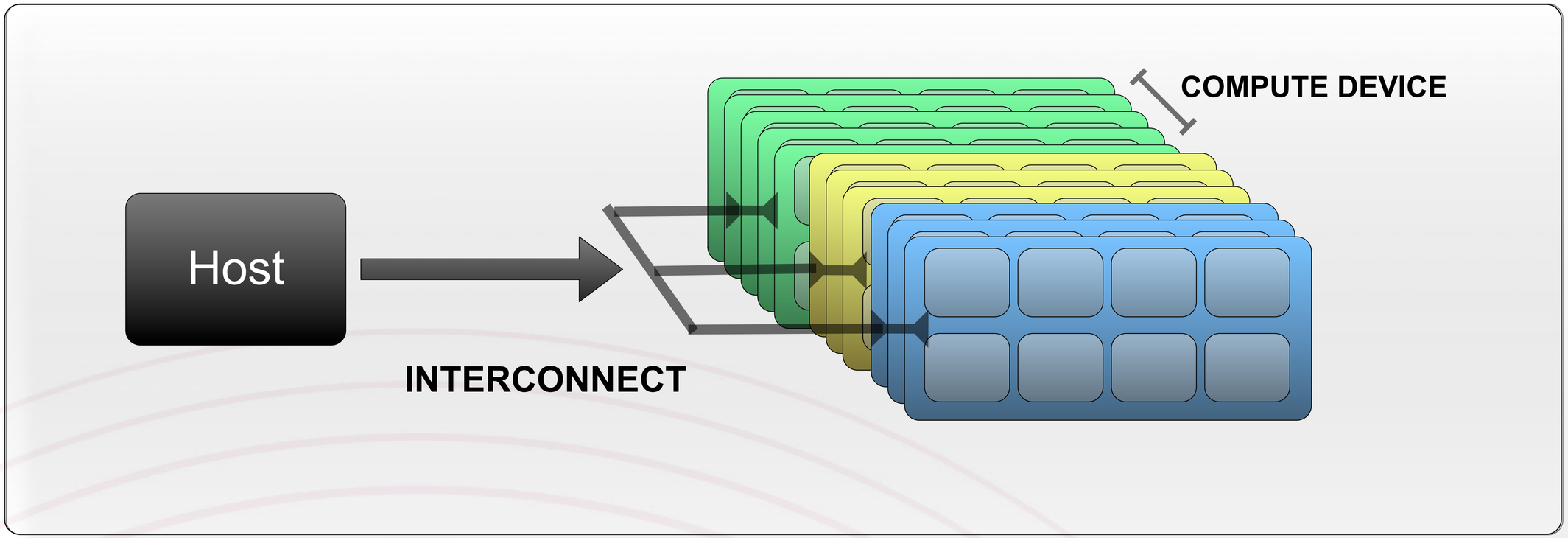
```
// Allocate input and output buffers, and fill the input with data
cl_mem input = clCreateBuffer(context,  CL_MEM_READ_ONLY,  sizeof(float) * count,
                              NULL, NULL);

// Create an output memory buffer for our results
cl_mem output = clCreateBuffer(context, CL_MEM_WRITE_ONLY, sizeof(float) * count,
                               NULL, NULL);

// Copy our host buffer of random values to the input device buffer
clEnqueueWriteBuffer(queue, input, CL_TRUE, 0, sizeof(float) * count, data, 0,
                     NULL, NULL);

// Get the maximum number of work items supported for this kernel on this device
size_t global = count; size_t local = 0;
clGetKernelWorkGroupInfo(kernel, device, CL_KERNEL_WORK_GROUP_SIZE, sizeof(int),
                         &local, NULL);
```

```
// Set the arguments to our kernel, and enqueue it for execution
clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
clSetKernelArg(kernel, 2, sizeof(unsigned int), &count);
clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global, &local, 0, NULL, NULL);

// Force the command queue to get processed, wait until all commands are complete
clFinish(queue);

// Read back the results
clEnqueueReadBuffer( queue, output, CL_TRUE, 0, sizeof(float) * count, results, 0,
                     NULL, NULL );

// Validate our results
int correct = 0;
for(i = 0; i < count; i++)
    correct += (results[i] == data[i] * data[i]) ? 1 : 0;

// Print a brief summary detailing the results
printf("Computed '%d/%d' correct values!\n", correct, count);
```

OpenCL Architecture

Conceptual models

- Platform model
- Execution model
- Memory model
- Programming model

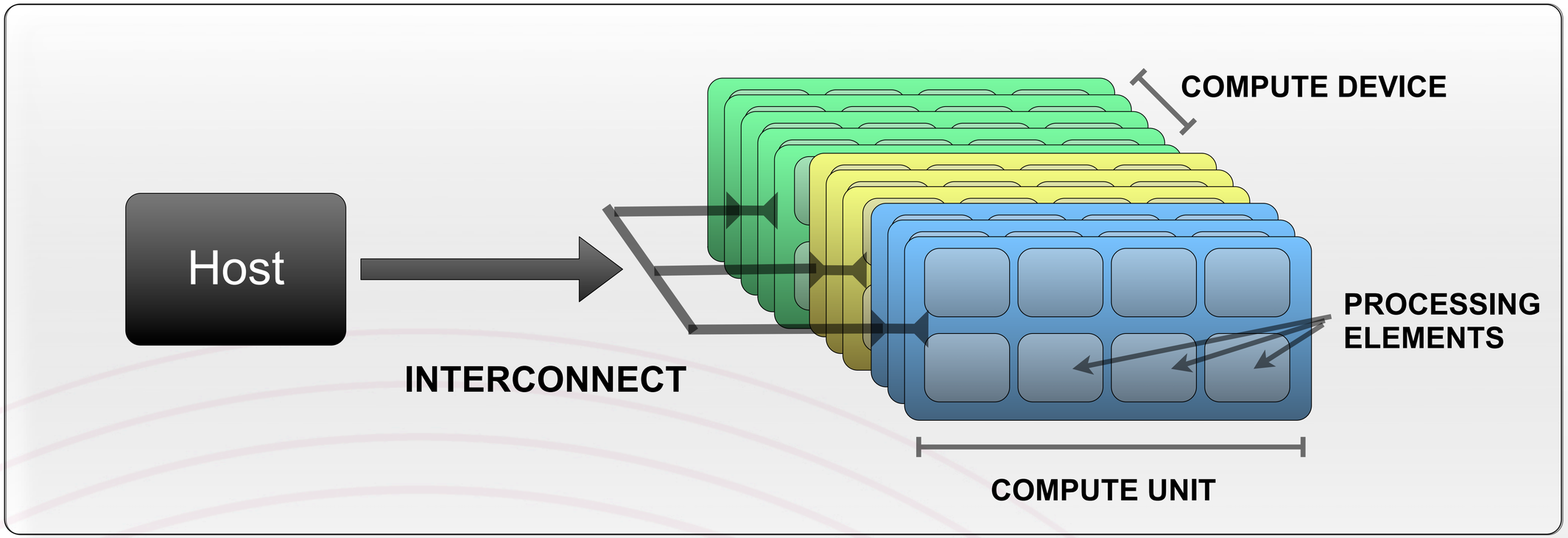


Platform model encapsulates compute resources

Hierarchy of compute units logically grouped together based on locality

One host connected to one or more compute devices

Compute device could be a CPU, GPU, or other processor.



Each compute device composed of one or more units

A compute unit may be a core, multi-processor, streaming mp, etc.

Each unit has one or more processing element(s)

Processing elements execute instructions together (eg. SIMD-style)

OpenCL API 1.1 Quick Reference Card - Page 1

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

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencl.

Querying Platform Info and Devices [4.1, 4.2]

cl_int **clGetPlatformIDs** (cl_uint *num_entries*, cl_platform_id **platforms*, cl_uint **num_platforms*)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*, cl_platform_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PLATFORM_{PROFILE, VERSION}, CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_int **clGetDeviceIDs** (cl_platform_id *platform*, cl_device_type *device_type*, cl_uint *num_entries*, cl_device_id **devices*, cl_uint **num_devices*)

device_type: CL_DEVICE_TYPE_{CPU, GPU}, CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue **clCreateCommandQueue** (cl_context *context*, cl_device_id *device*, cl_command_queue_properties *properties*, cl_int **errcode_ret*)

properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

cl_int **clRetainCommandQueue** (cl_command_queue *command_queue*)

cl_int **clReleaseCommandQueue** (cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (cl_command_queue *command_queue*, cl_command_queue_info *param_name*, size_t *param_value_size*, void **param_value*, size_t **param_value_size_ret*)

param_name: CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE, CL_QUEUE_REFERENCE_COUNT, CL_QUEUE_PROPERTIES

Buffer Objects

Elements of a buffer object can be a scalar or vecto… a user-defined structure. Elements are stored sequ… are accessed using a pointer by a kernel executing on a dev… Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem **clCreateBuffer** (cl_context *context*, cl_mem_flags *flags*, size_t *size*, void **host_ptr*, cl_int **errcode_ret*)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*, cl_mem_flags *flags*, cl_buffer_create_type *buffer_create_type*, const void **buffer_create_info*, cl_int **errcode_ret*)

flags for **clCreateBuffer** and **clCreateSubBuffer**: CL_MEM_READ_WRITE,

size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*, size_t *host_row_pitch*, size_t *host_slice_pitch*, void **ptr*, cl_uint *num_events_in_wait_list*, const 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*[3], const size_t *host_origin*[3], const size_t *region*[3], size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*, size_t *host_row_pitch*, size_t *host_slice_pitch*, void **ptr*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*)

size_t *offset*, size_t *cb*, cl_uint *num_events_in_wait_list*, const cl_event **event_wait_list*, cl_event **event*, cl_int **errcode_ret*)

Map Buffer Objects [5.4.1-2]

cl_int **clRetainMemObject** (cl_mem *memobj*)

cl_int **clReleaseMemObject** (cl_mem *memobj*)

cl_int **clSetMemObjectDestructorCallback** (cl_mem *memobj*, void (CL_CALLBACK **pfn_notify*) (cl_mem *memobj*, void **user_data*), void **user_data*)

OpenCL API 1.1 Quick Reference Card - Page 1

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

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencl.

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue **clCreateCommandQueue** (
cl_context *context*, cl_device_id *device*,
cl_command_queue_properties *properties*,
cl_int *errcode_ret*)

properties: CL_QUEUE_PROFILING_ENABLE,
CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

cl_int **clRetainCommandQueue** (
cl_command_queue *command_queue*)

cl_int **clReleaseCommandQueue** (
cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (
cl_command_queue *command_queue*,
cl_command_queue_info *param_name*,
size_t *param_value_size*, void *param_value*,
size_t *param_value_size_ret*)

param_name: CL_QUEUE_CONTEXT,
CL_QUEUE_DEVICE,
CL_QUEUE_REFERENCE_COUNT,
CL_QUEUE_PROPERTIES

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

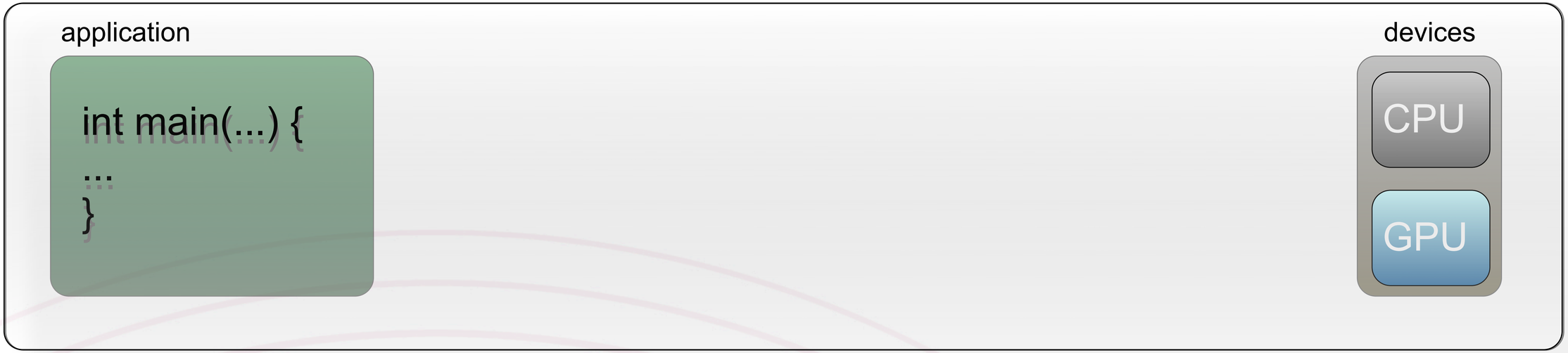
cl_mem **clCreateBuffer** (cl_context *context*,
cl_mem_flags *flags*, size_t *size*, void *host_ptr*,
cl_int *errcode_ret*)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*,
cl_mem_flags *flags*,
cl_buffer_create_type *buffer_create_type*,
const void *buffer_create_info*, cl_int *errcode_ret*)

flags for **clCreateBuffer** and **clCreateSubBuffer**:
CL_MEM_READ_WRITE,

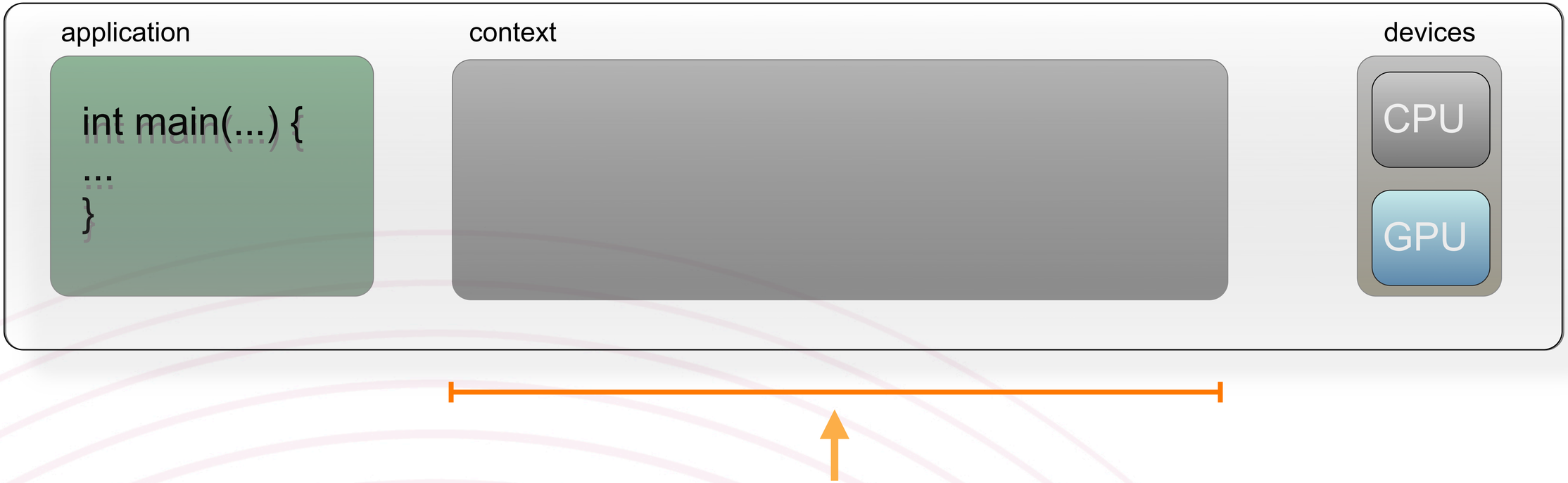
cl_int **clGetDeviceInfo** (cl_device_id *device*,
cl_device_info *param_name*, size_t *param_value_size*,
void *param_value*, size_t *param_value_size_ret*)

param_name: CL_DEVICE_TYPE,
CL_DEVICE_VENDOR_ID,
CL_DEVICE_MAX_COMPUTE_UNITS,
CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES},
CL_DEVICE_MAX_WORK_GROUP_SIZE,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_CHAR,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_SHORT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_INT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_LONG,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_FLOAT,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_DOUBLE,
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_HALF,
CL_DEVICE_MAX_CLOCK_FREQUENCY,
CL_DEVICE_ADDRESS_BITS,
CL_DEVICE_MAX_MEM_ALLOC_SIZE,
CL_DEVICE_IMAGE_SUPPORT,
CL_DEVICE_MAX_{READ, WRITE}_IMAGE_ARGS,



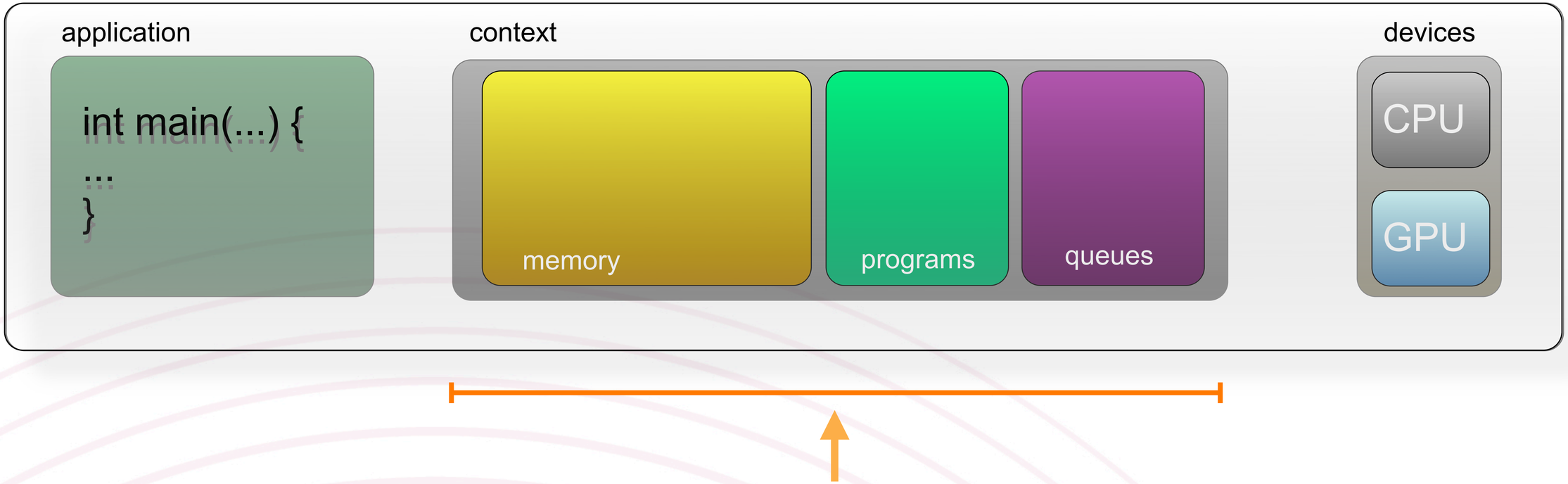
Application needs access to compute resources in system

Execution model defines the interaction between the host application and a device



Context encapsulates resources into a logical grouping

Think of this as a container of compute resources for your application

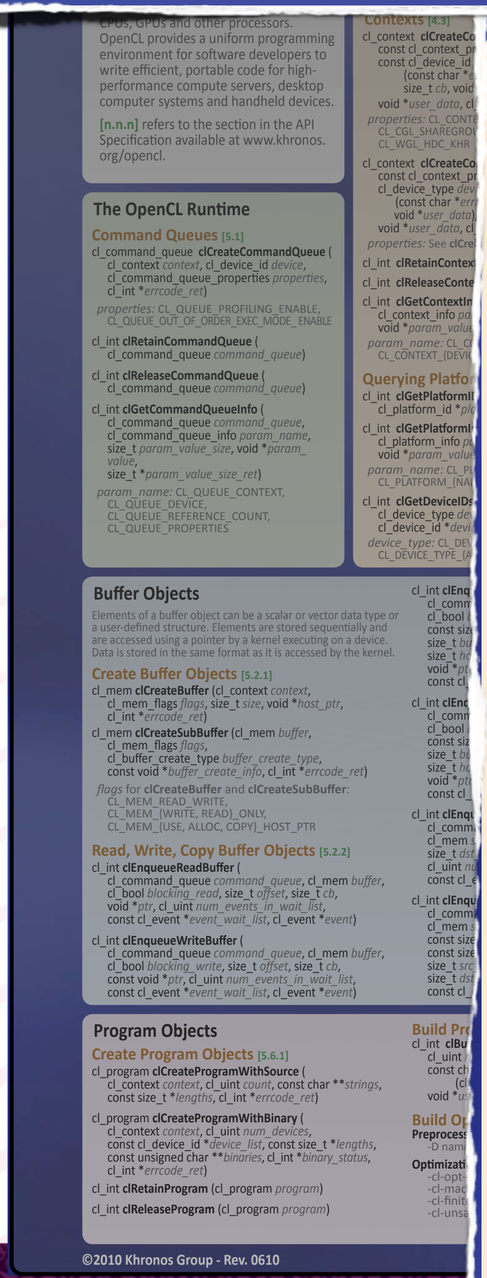


Memory, programs, and queues are owned by the context

Provides a container that localises allocations into a central store

Also, enables transparent sharing with OpenGL / OpenGL-ES / DirectX

OpenCL API 1.1 Quick Reference Card - Page 1


CPUs, GPUs and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems and handheld devices.

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencl.

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue clCreateCommandQueue (
cl_context context, cl_device_id device,
cl_command_queue_properties properties,
cl_int *errcode_ret)

properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

cl_int clRetainCommandQueue (
cl_command_queue command_queue)

cl_int clReleaseCommandQueue (
cl_command_queue command_queue)

cl_int clGetCommandQueueInfo (
cl_command_queue command_queue,
cl_command_queue_info param_name,
size_t param_value_size, void *param_value,
size_t *param_value_size_ret)

param_name: CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE, CL_QUEUE_REFERENCE_COUNT, CL_QUEUE_PROPERTIES

Contexts [4.3]

cl_context clCreateCo
const cl_context_pr
const cl_device_id
(const char *e
size_t cb, void
void *user_data, cl

properties: CL_CONTE
CL_CGL_SHAREGRO
CL_WGL_HDC_KHR

cl_context clCreateCo
const cl_context_pr
cl_device_type dev
(const char *err
void *user_data),
void *user_data, cl

properties: See clCre

cl_int clRetainContex

cl_int clReleaseConte

cl_int clGetContextIn
cl_context_info pa
void *param_valu

param_name: CL_C
CL_CONTEXT_{DEVI

Querying Platfor

cl_int clGetPlatformI
cl_platform_id *pl

cl_int clGetPlatformI
cl_platform_info pa
void *param_value

param_name: CL_PL
CL_PLATFORM_{NA

cl_int clGetDeviceIDs
cl_device_type de
cl_device_id *devi

device_type: CL_DE
CL_DEVICE_TYPE_{A

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem clCreateBuffer (cl_context context,
cl_mem_flags flags, size_t size, void *host_ptr,
cl_int *errcode_ret)

cl_mem clCreateSubBuffer (cl_mem buffer,
cl_mem_flags flags,
cl_buffer_create_type buffer_create_type,
const void *buffer_create_info, cl_int *errcode_ret)

flags for clCreateBuffer and clCreateSubBuffer: CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int clEnqueueReadBuffer (
cl_command_queue command_queue, cl_mem buffer,
cl_bool blocking_read, size_t offset, size_t cb,
void *ptr, cl_uint num_events_in_wait_list,
const cl_event *event_wait_list, cl_event *event)

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

cl_int clEnq
cl_comm
cl_bool
const siz
size_t bu
size_t ho
void *pt
const cl_

cl_int clEnq
cl_comm
cl_bool
const siz
size_t bu
size_t ho
void *pt
const cl_

cl_int clEnqu
cl_comm
cl_mem s
size_t dst
cl_uint nu
const cl_e

cl_int clEnqu
cl_comm
cl_mem s
const size
const size
size_t src
size_t dst
const cl_

Program Objects

Create Program Objects [5.6.1]

cl_program clCreateProgramWithSource (
cl_context context, cl_uint count, const char **strings,
const size_t *lengths, cl_int *errcode_ret)

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

cl_int clRetainProgram (cl_program program)

cl_int clReleaseProgram (cl_program program)

Build Pro

cl_int clBu
cl_uint
const cl
(cl
void *u

Build Op

Preprocess
-D nam

Optimizati
-cl-opt
-cl-mad
-cl-finit
-cl-unsa

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allo information, and to create OpenCL contexts using one or more devices.

Contexts [4.3]

cl_context **clCreateContext** (
const cl_context_properties *properties*, cl_uint *num_devices*,
const cl_device_id *devices*, void (CL_CALLBACK*pfn_notify*)
(const char *errinfo*, const void *private_info*,
size_t *cb*, void *user_data*),
void *user_data*, cl_int *errcode_ret*)

properties: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR, CL_WGL_HDC_KHR

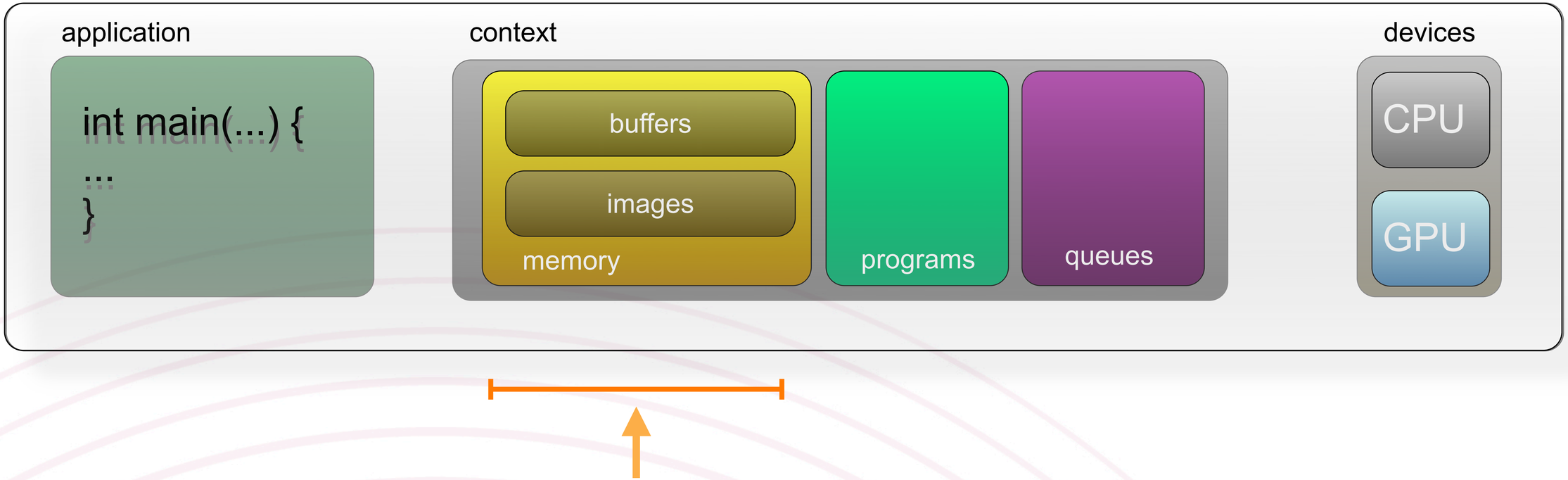
cl_context **clCreateContextFromType** (
const cl_context_properties *properties*,
cl_device_type *device_type*, void (CL_CALLBACK *pfn_notify*)
(const char *errinfo*, const void *private_info*, size_t *cb*,
void *user_data*),
void *user_data*, cl_int *errcode_ret*)

properties: See **clCreateContext**

cl_int **clRetainContext** (cl_context *context*)

cl_int **clReleaseContext** (cl_context *context*)

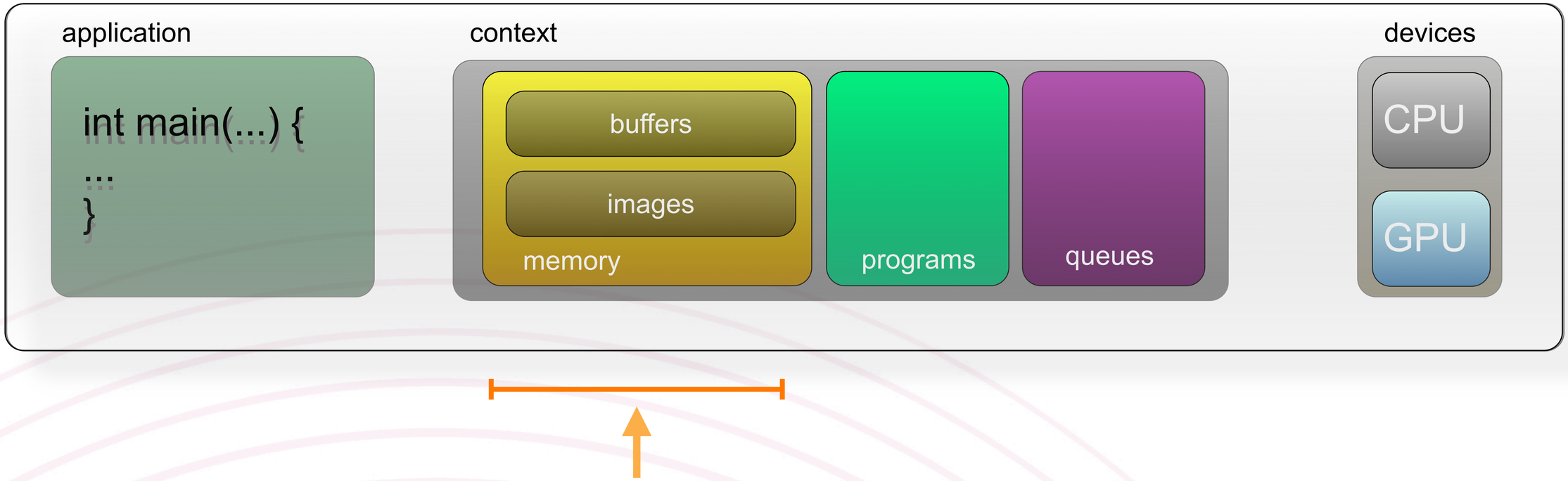
cl_int **clGetContextInfo** (cl_context *context*,
cl_context_info *param_name*, size_t *param_value_size*,
void *param_value*, size_t *param_value_size_ret*)



Memory allocations can be raw bytes or formatted images

Buffers are basically the same as a malloc in C99 -- contiguous allocation of bytes

Images provide a direct path to texture hardware and support sampling, filtering, etc.



Memory is accessible for all devices in context

However, synchronisation is **required** to get expected results!

OpenCL API 1.1 Quick Reference Card - Page 1


environment for software developers
write efficient, portable code for high
performance compute servers, deskt
computer systems and handheld devi

[n.n.n] refers to the section in the API
Specification available at www.khrono
org/opencl.

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue clCreateCommandQue
cl_context context, cl_device_id device,
cl_command_queue_properties propert
cl_int *errcode_ret)

properties: CL_QUEUE_PROFILING_ENABL
CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_EN

cl_int clRetainCommandQueue (
cl_command_queue command_queue)

cl_int clReleaseCommandQueue (
cl_command_queue command_queue)

cl_int clGetCommandQueueInfo (
cl_command_queue command_queue,
cl_command_queue_info param_name,
size_t param_value_size, void *param_
value,
size_t *param_value_size_ret)

param_name: CL_QUEUE_CONTEXT,
CL_QUEUE_DEVICE,
CL_QUEUE_REFERENCE_COUNT,
CL_QUEUE_PROPERTIES

Buffer Objects

Elements of a buffer object can be a scalar or
a user-defined structure. Elements are stored
are accessed using a pointer by a kernel exec
Data is stored in the same format as it is acce

Create Buffer Objects [5.2.1]

cl_mem clCreateBuffer (cl_context context
cl_mem_flags flags, size_t size, void *ho
cl_int *errcode_ret)

cl_mem clCreateSubBuffer (cl_mem buffer,
cl_mem_flags flags,
cl_buffer_create_type buffer_create_typ
const void *buffer_create_info, cl_int *er

flags for clCreateBuffer and clCreateSubB
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects

cl_int clEnqueueReadBuffer (
cl_command_queue command_queue, c
cl_bool blocking_read, size_t offset, size_
void *ptr, cl_uint num_events_in_wait_li
const cl_event *event_wait_list, cl_even

cl_int clEnqueueWriteBuffer (
cl_command_queue command_queue,
cl_bool blocking_write, size_t offset, siz
const void *ptr, cl_uint num_events_in_
const cl_event *event_wait_list, cl_ever

Program Objects

Create Program Objects [5.6.1]

cl_program clCreateProgramWithSource (
cl_context context, cl_uint count, const
const size_t *lengths, cl_int *errcode_r

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

cl_int clRetainProgram (cl_program progr

cl_int clReleaseProgram (cl_program prog

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem **clCreateBuffer** (cl_context *context*,
cl_mem_flags *flags*, size_t *size*, void *host_ptr,
cl_int *errcode_ret)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*,
cl_mem_flags *flags*,
cl_buffer_create_type *buffer_create_type*,
const void *buffer_create_info, cl_int *errcode_ret)

flags for **clCreateBuffer** and **clCreateSubBuffer**:
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int **clEnqueueReadBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, size_t *offset*, size_t *cb*,
void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

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

cl_int **clEnqueueReadBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, const size_t *buffer_origin*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void *ptr, cl_uint *num_events_in_wait_list*,
const 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*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void *ptr, cl_uint *num_events_in_wait_list*,
const 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 *cb*,
cl_uint *num_events_in_wait_list*,
const 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*[3], const size_t *dst_origin*[3],
const size_t *region*[3], size_t *src_row_pitch*,
size_t *src_slice_pitch*, size_t *dst_row_pitch*,
size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

Image Objects

Create Image Objects [5.3.1]

cl_mem **clCreateImage2D** (cl_context *context*,
cl_mem_flags *flags*, const cl_image_format **image_format*,
size_t *image_width*, size_t *image_height*,
size_t *image_row_pitch*, void **host_ptr*, cl_int **errcode_ret*)

flags: (also for **clCreateImage3D**, **clGetSupportedImageFormats**)
CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

cl_mem **clCreateImage3D** (cl_context *context*,
cl_mem_flags *flags*, const cl_image_format **image_format*,
size_t *image_width*, size_t *image_height*, size_t *image_depth*, size_t *image_row_pitch*, size_t *image_slice_pitch*,
void **host_ptr*, cl_int **errcode_ret*)

flags: See **clCreateImage2D**

Query List of Supported Image Formats [5.3.2]

cl_int **clGetSupportedImageFormats** (cl_context *context*,
cl_mem_flags *flags*, cl_mem_object_type *image_type*,
cl_uint *num_entries*, cl_image_format **image_formats*,
cl_uint **num_image_formats*)

flags: See **clCreateImage2D**

Copy Between Image, Buffer Objects [5.3.4]

cl_int **clEnqueueCopyImageToBuffer** (
cl_command_queue *command_queue*, cl_mem *src_image*,
cl_mem *dst_buffer*, const size_t *src_origin*[3],
const size_t *region*[3], size_t *dst_offset*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Read, Write, Copy Image Objects [5.3.3]

cl_int **clEnqueueReadImage** (
cl_command_queue *command_queue*, cl_mem *image*,
cl_bool *blocking_read*, const size_t *origin*[3],
const size_t *region*[3], size_t *row_pitch*,
size_t *slice_pitch*, void **ptr*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

cl_int **clEnqueueWriteImage** (
cl_command_queue *command_queue*,
cl_mem *image*, cl_bool *blocking_write*,
const size_t *origin*[3], const size_t *region*[3],
size_t *input_row_pitch*, size_t *input_slice_pitch*,
const void **ptr*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

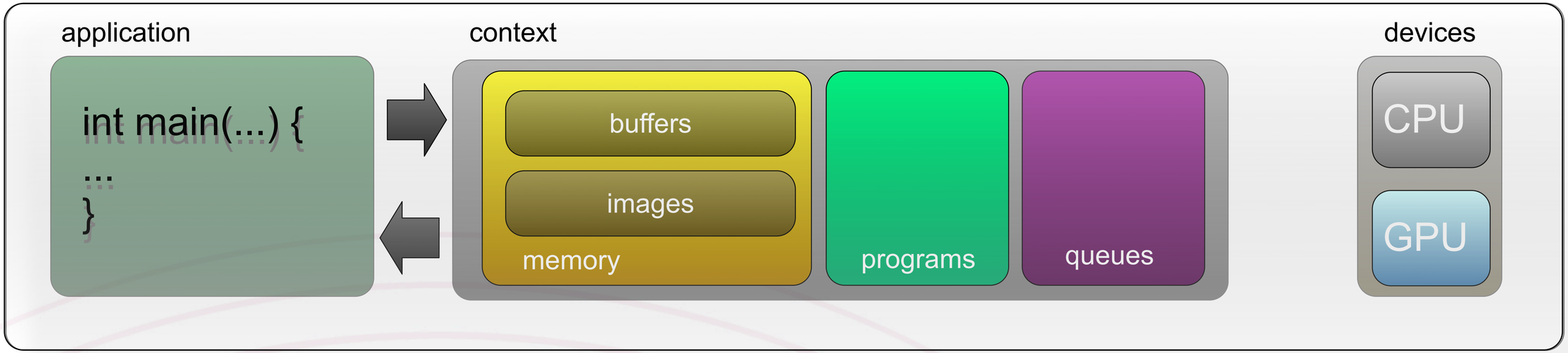
cl_int **clEnqueueCopyImage** (
cl_command_queue *command_queue*,
cl_mem *src_image*, cl_mem *dst_image*,
const size_t *src_origin*[3], const size_t *dst_origin*[3],
const size_t *region*[3], cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Query Image Objects [5.3.6]

cl_int **clGetMemObjectInfo** (cl_mem *memobj*,
cl_mem_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

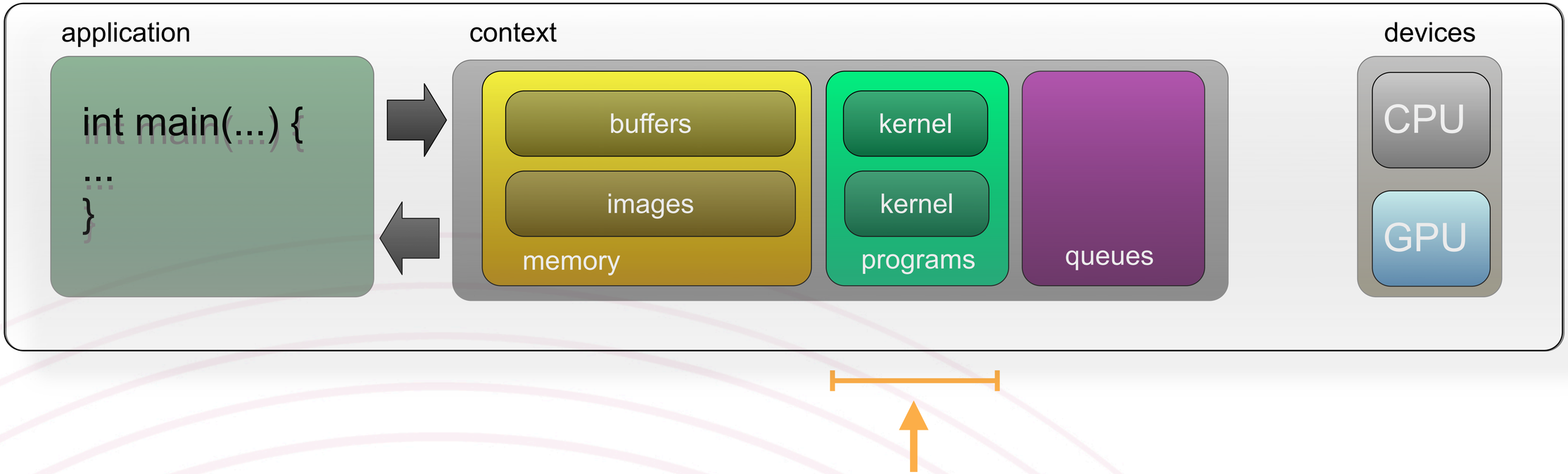
param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
CL_MEM_{MAP, REFERENCE}_COUNT,
CL_MEM_{CONTEXT, OFFSET},
CL_MEM_ASSOCIATED_MEMOBJECT

cl_int **clGetImageInfo** (cl_mem *image*,



Application uses context for moving data to/from host

All data transfers between host and device must be done explicitly!



Programs contain compiled kernel objects

Each program must be compiled for every type of device and/or instruction set

OpenCL API 1.1 Quick Reference Card - Page 1

Program Objects

Create Program Objects [5.6.1]

cl_program **clCreateProgramWithSource** (
    cl_context *context*, cl_uint *count*, const char **_strings_,
    const size_t *_lengths_, cl_int *_errcode_ret_)

cl_program **clCreateProgramWithBinary** (
    cl_context *context*, cl_uint *num_devices*,
    const cl_device_id *_device_list_, const size_t *_lengths_,
    const unsigned char **_binaries_, cl_int *_binary_status_,
    cl_int *_errcode_ret_)

cl_int **clRetainProgram** (cl_program *program*)

cl_int **clReleaseProgram** (cl_program *program*)

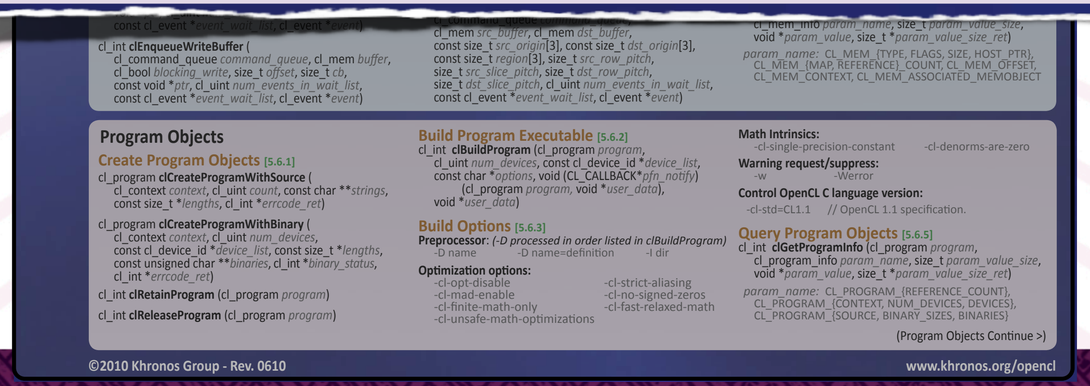
Build Program Executable [5.6.2]

cl_int **clBuildProgram** (cl_program *program*,
    cl_uint *num_devices*, const cl_device_id *_device_list_,
    const char *_options_, void (CL_CALLBACK*_pfn_notify_)
        (cl_program *program,* void *_user_data_),
    void *_user_data_)

Build Options [5.6.3]

Preprocessor: *(-D processed in order listed in clBuildProgram)*
    -D name -D name=definition -I dir

Optimization options:
    -cl-opt-disable -cl-strict-aliasing
    -cl-mad-enable -cl-no-signed-zeros
    -cl-finite-math-only -cl-fast-relaxed-math
    -cl-unsafe-math-optimizations



size_t *param_value_size_ret)
param_name: CL_PROGRAM_BUILD_{STATUS, OPTIONS, LOG}

Unload the OpenCL Compiler [5.6.4]
cl_int **clUnloadCompiler** (void)

Supported Data Types

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_half	16-bit float (for storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	void	void

Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
charn	cl_charn	8-bit signed
ucharn	cl_ucharn	8-bit unsigned
shortn	cl_shortn	16-bit signed
ushortn	cl_ushortn	16-bit unsigned
intn	cl_intn	32-bit signed
uintn	cl_uintn	32-bit unsigned
longn	cl_longn	64-bit signed
ulongn	cl_ulongn	64-bt unsigned
floatn	cl_floatn	32-bit float

Other Built-in Data Types [6.1.3]

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
sampler_t	sampler handle
event_t	event handle

Reserved Data Types [6.1.4]

OpenCL Type	Description
booln	boolean vector
double, doublen OPTIONAL	64-bit float, vector
halfn	16-bit, vector
quad, quadn	128-bit float, vector
complex half, complex halfn imaginary half, imaginary halfn	16-bit complex, vector
complex float, complex floatn imaginary float, imaginary floatn	32-bit complex, vector
complex double, complex doublen imaginary double, imaginary doublen	64-bit complex, vector
complex quad, complex quadn imaginary quad, imaginary quadn	128-bit complex, vector
floatnxm	n*m matrix of 32-bit floats
doublenxm	n*m matrix of 64-bit floats
long double, long doublen	64 - 128-bit float, vector
long long, long longnb	128-bit signed
unsigned long long, ulong long, ulong longn	128-bit unsigned

const char *kernel_name, cl_int *errcode_ret)
cl_int **clCreateKernelsInProgram** (cl_program pro
cl_uint *num_kernels*, cl_kernel **kernels*,
cl_uint **num_kernels_ret*)
cl_int **clRetainKernel** (cl_kernel *kernel*)
cl_int **clReleaseKernel** (cl_kernel *kernel*)

Kernel Args. & Object Queries [5.7.2, 5.
cl_int **clSetKernelArg** (cl_kernel *kernel*, cl_uint *arg*
size_t *arg_size*, const void **arg_value*)

cl_int **clGetKernelInfo** (cl_kernel *kernel*,
cl_kernel_info *param_name*, size_t *param_val*
void **param_value*, size_t **param_value_size*
param_name: CL_KERNEL_FUNCTION_NAME,
CL_KERNEL_NUM_ARGS, CL_KERNEL_REFERENCE
CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM

cl_int **clGetKernelWorkGroupInfo** (
cl_kernel *kernel*, cl_device_id *device*,
cl_kernel_work_group_info *param_name*,
size_t *param_value_size*, void **param_value*,
size_t **param_value_size_ret*)
param_name: CL_KERNEL_WORK_GROUP_SIZE
CL_KERNEL_COMPILE_WORK_GROUP_SIZE,
CL_KERNEL_{LOCAL, PRIVATE}_MEM_SIZE,
CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_

Execute Kernels [5.8]
cl_int **clEnqueueNDRangeKernel** (
cl_command_queue *command_queue*,
cl_kernel *kernel*, cl_uint *work_dim*,
const size_t **global_work_offset*,
const size_t **global_work_size*,
const size_t **local_work_size*,
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **eve
cl_int **clEnqueueTask** (
cl_command_queue *command_queue*, cl_ker
kernel, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **eve
cl_int **clEnqueueNativeKernel** (cl_command_que
command_queue, void (**user_func*)(void *),
void **args*, size_t *cb_args*, cl_uint *num_mem*
const cl_mem **mem_list*, const void ***args_*
cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **ev

Vector Component Addressing [6.

Vector Components

	0	1	2	3
float2 v;	v.x, v.s0	v.y, v.s1		
float3 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2	
float4 v;	v.x, v.s0	v.y, v.s1	v.z, v.s2	v.w, v.s3
float8 v;	v.s0	v.s1	v.s2	v.s3
float16 v;	v.s0	v.s1	v.s2	v.s3

Vector Addressing Equivalencies
Numeric indices are preceded by the letter s or S, e

	v.lo	v.hi	v.odd	v.
float2	v.x, v.s0	v.y, v.s1	v.y, v.s1	v.x
float3	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s0
float4	v.s01, v.xy	v.s23, v.zw	v.s13, v.yw	v.s0

Conversions & Type Casting Example
T a = (*T*)*b*; *// Scalar to scalar, or scalar to vector*
T a = convert_*T*(*b*);
T a = convert_*T_R*(*b*);
T a = as_*T*(*b*);

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

\+ - * % / -- ++ == !
~ ^ > < >= <= | ! &
?: >> << , = *op*= *sizeof*

Kernel and Event Objects

Create Kernel Objects [5.7.1]

cl_kernel **clCreateKernel** (cl_program *program*,
const char **kernel_name*, cl_int **errcode_ret*)

cl_int **clCreateKernelsInProgram** (cl_program *program*,
cl_uint *num_kernels*, cl_kernel **kernels*,
cl_uint **num_kernels_ret*)

cl_int **clRetainKernel** (cl_kernel *kernel*)

cl_int **clReleaseKernel** (cl_kernel *kernel*)

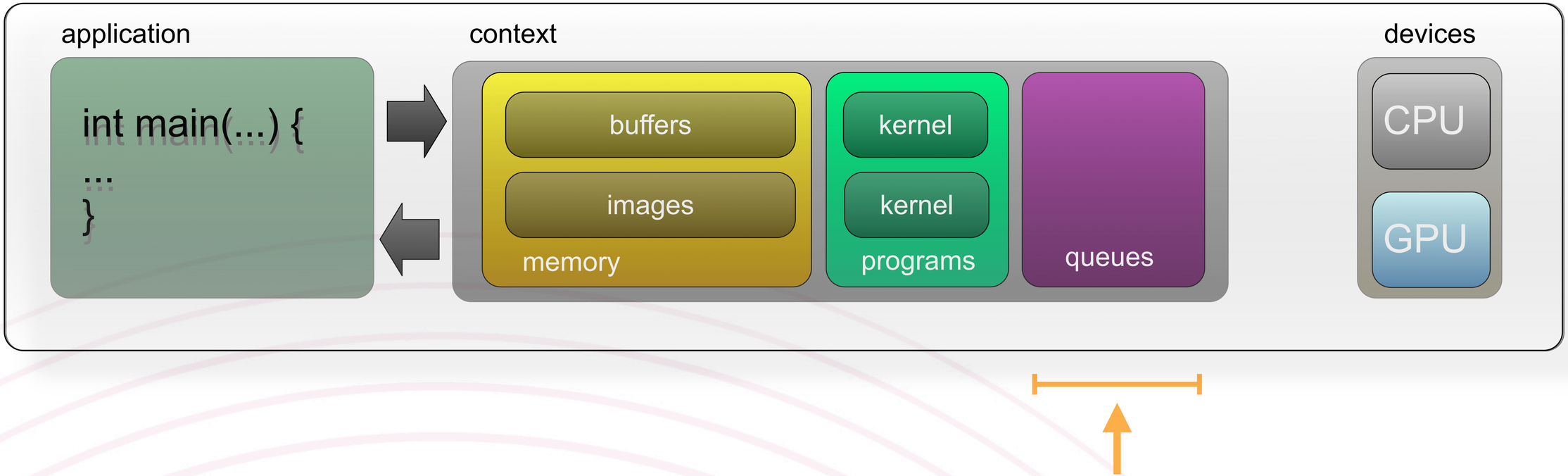
Kernel Args. & Object Queries [5.7.2, 5.7.3]

cl_int **clSetKernelArg** (cl_kernel *kernel*, cl_uint *arg_index*,
size_t *arg_size*, const void **arg_value*)

cl_int **clGetKernelInfo** (cl_kernel *kernel*,
cl_kernel_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

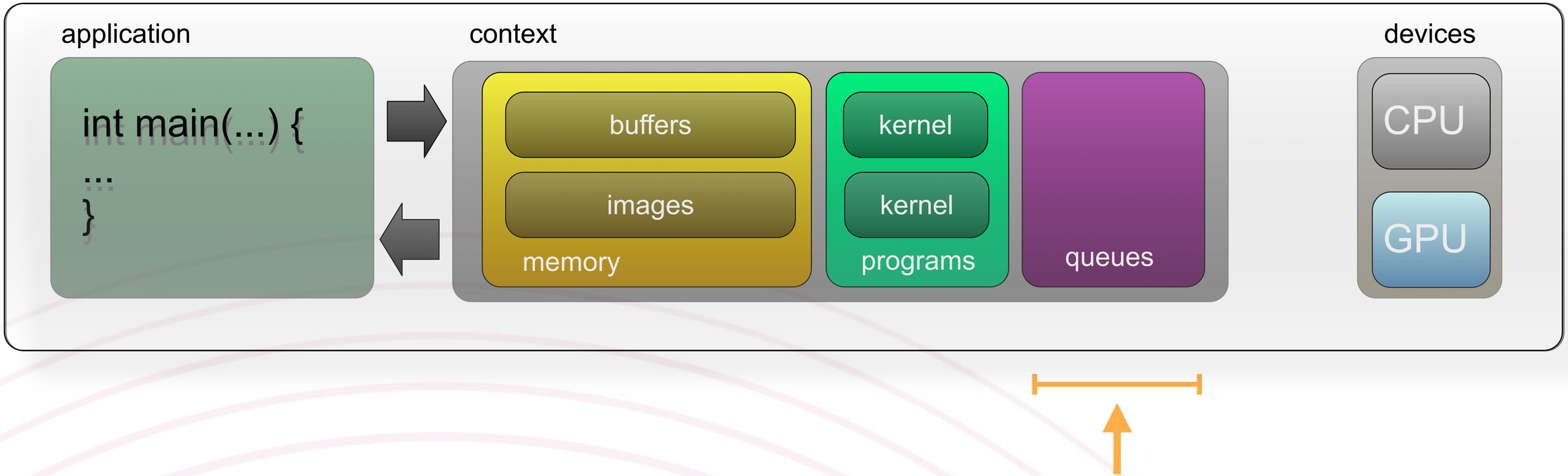
param_name: CL_KERNEL_FUNCTION_NAME,
CL_KERNEL_NUM_ARGS, CL_KERNEL_REFERENCE_COUNT,
CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM

cl_int **clGetKernelWorkGroupInfo** (



Command queues provide asynchronous execution

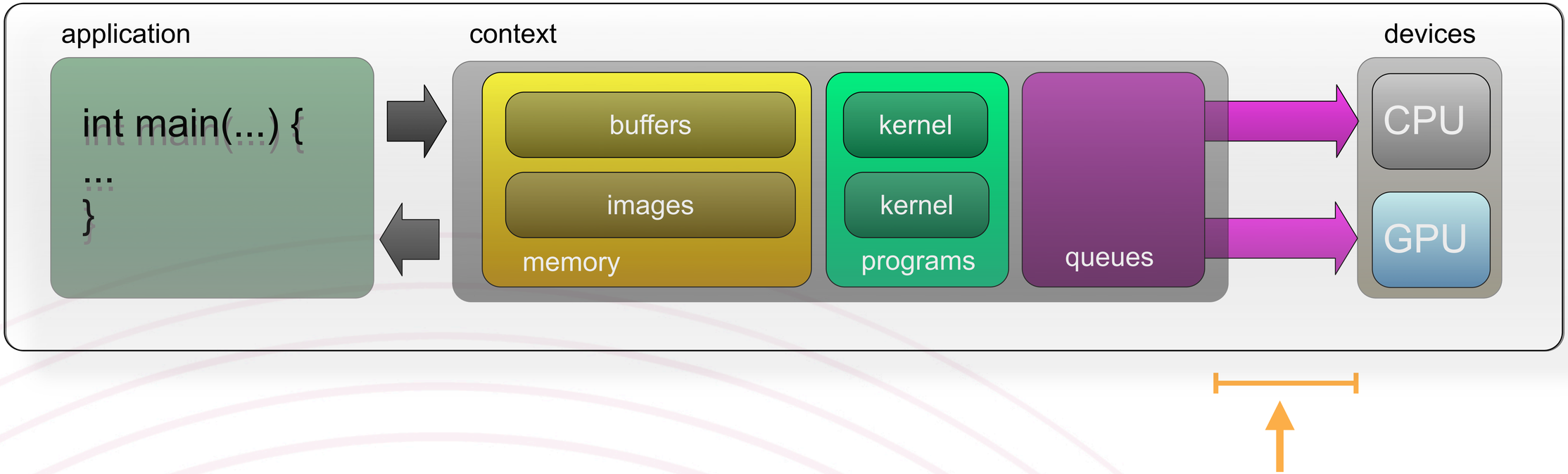
Enqueue commands onto a command queue to do work (sometime later)



Queues can be processed in-order or out-of-order

In-order queues are default -- commands run when preceding entries are complete

Out-of-order queues require explicit synchronisation via events and wait-lists



Queues are directly associated with a specific device

Submit commands to the associated queue to do work on a particular device

OpenCL API 1.1 Quick Reference Card - Page 1

CPUs, GPUs and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems and handheld devices.

[n.n.n] refers to the section in the API Specification available at www.khronos.org/opencl.

The OpenCL Runtime

Command Queues [5.1]

cl_command_queue **clCreateCommandQueue** (
cl_context *context*, cl_device_id *device*,
cl_command_queue_properties *properties*,
cl_int **errcode_ret*)

properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

cl_int **clRetainCommandQueue** (
cl_command_queue *command_queue*)

cl_int **clReleaseCommandQueue** (
cl_command_queue *command_queue*)

cl_int **clGetCommandQueueInfo** (
cl_command_queue *command_queue*,
cl_command_queue_info *param_name*,
size_t *param_value_size*, void **param_value*,
size_t **param_value_size_ret*)

param_name: CL_QUEUE_CONTEXT, CL_QUEUE_DEVICE, CL_QUEUE_REFERENCE_COUNT, CL_QUEUE_PROPERTIES

Contexts [4.3]

cl_context **clCreateContext** (
const cl_context_properties **properties*, cl_uint *num_devices*,
const cl_device_id **devices*, void (CL_CALLBACK**pfn_notify*)
(const char **errinfo*, const void **private_info*,
size_t *cb*, void **user_data*),
void **user_data*, cl_int **errcode_ret*)

properties: CL_CONTEXT_PLATFORM, CL_GL_CONTEXT_KHR, CL_CGL_SHAREGROUP_KHR, CL_{EGL, GLX}_DISPLAY_KHR, CL_WGL_HDC_KHR

cl_context **clCreateContextFromType** (
const cl_context_properties **properties*,
cl_device_type *device_type*, void (CL_CALLBACK **pfn_notify*)
(const char **errinfo*, const void **private_info*, size_t *cb*,
void **user_data*),
void **user_data*, cl_int **errcode_ret*)

properties: See **clCreateContext**

cl_int **clRetainContext** (cl_context *context*)

cl_int **clReleaseContext** (cl_context *context*)

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*)

param_name: CL_CONTEXT_REFERENCE_COUNT, CL_CONTEXT_{DEVICES, PROPERTIES}, CL_CONTEXT_NUM_DEVICES

Querying Platform Info and Devices [4.1, 4.2]

cl_int **clGetPlatformIDs** (cl_uint *num_entries*,
cl_platform_id **platforms*, cl_uint **num_platforms*)

cl_int **clGetPlatformInfo** (cl_platform_id *platform*,
cl_platform_info *param_name*, size_t *param_value_size*,
void **param_value*, size_t **param_value_size_ret*)

param_name: CL_PLATFORM_{PROFILE, VERSION}, CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_int **clGetDeviceIDs** (cl_platform_id *platform*,
cl_device_type *device_type*, cl_uint *num_entries*,
cl_device_id **devices*, cl_uint **num_devices*)

device_type: CL_DEVICE_TYPE_{CPU, GPU}, CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem **clCreateBuffer** (cl_context *context*,
cl_mem_flags *flags*, size_t *size*, void **host_ptr*,
cl_int **errcode_ret*)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*,
cl_mem_flags *flags*,
cl_buffer_create_type *buffer_create_type*,
const void **buffer_create_info*, cl_int **errcode_ret*)

flags for **clCreateBuffer** and **clCreateSubBuffer**:
CL_MEM_READ_WRITE, CL_MEM_{WRITE, READ}_ONLY, CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int **clEnqueueReadBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, size_t *offset*, size_t *cb*,
void **ptr*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

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

cl_int **clEnqueueReadBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, const size_t *buffer_origin*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void **ptr*, cl_uint *num_events_in_wait_list*,
const 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*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void **ptr*, cl_uint *num_events_in_wait_list*,
const 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 *cb*,
cl_uint *num_events_in_wait_list*,
const 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*[3], const size_t *dst_origin*[3],
const size_t *region*[3], size_t *src_row_pitch*,
size_t *src_slice_pitch*, size_t *dst_row_pitch*,
size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*,
const cl_event **event_wait_list*, cl_event **event*)

Program Objects

Create Program Objects [5.6.1]

cl_program **clCreateProgramWithSource** (
cl_context *context*, cl_uint *count*, const char ***strings*,
const size_t **lengths*, cl_int **errcode_ret*)

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

cl_int **clRetainProgram** (cl_program *program*)

cl_int **clReleaseProgram** (cl_program *program*)

Build Program Executable [5.6.2]

cl_int **clBuildProgram** (cl_program *program*,
cl_uint *num_devices*, const cl_device_id **device_list*,
const char **options*, void (CL_CALLBACK**pfn_notify*)
(cl_program *program*, void **user_data*),
void **user_data*)

Build Options [5.6.3]

Preprocessor: *(-D processed in order listed in clBuildProgram)*
-D name -D name=definition -I dir

Optimization options:
-cl-opt-disable
-cl-mad-enable
-cl-finite-math-only
-cl-unsafe-math-optimizations
-cl-strict-aliasing
-cl-no-signed-zeros
-cl-fast-relaxed-math

The OpenCL Runtime

Command Queues [5.1]

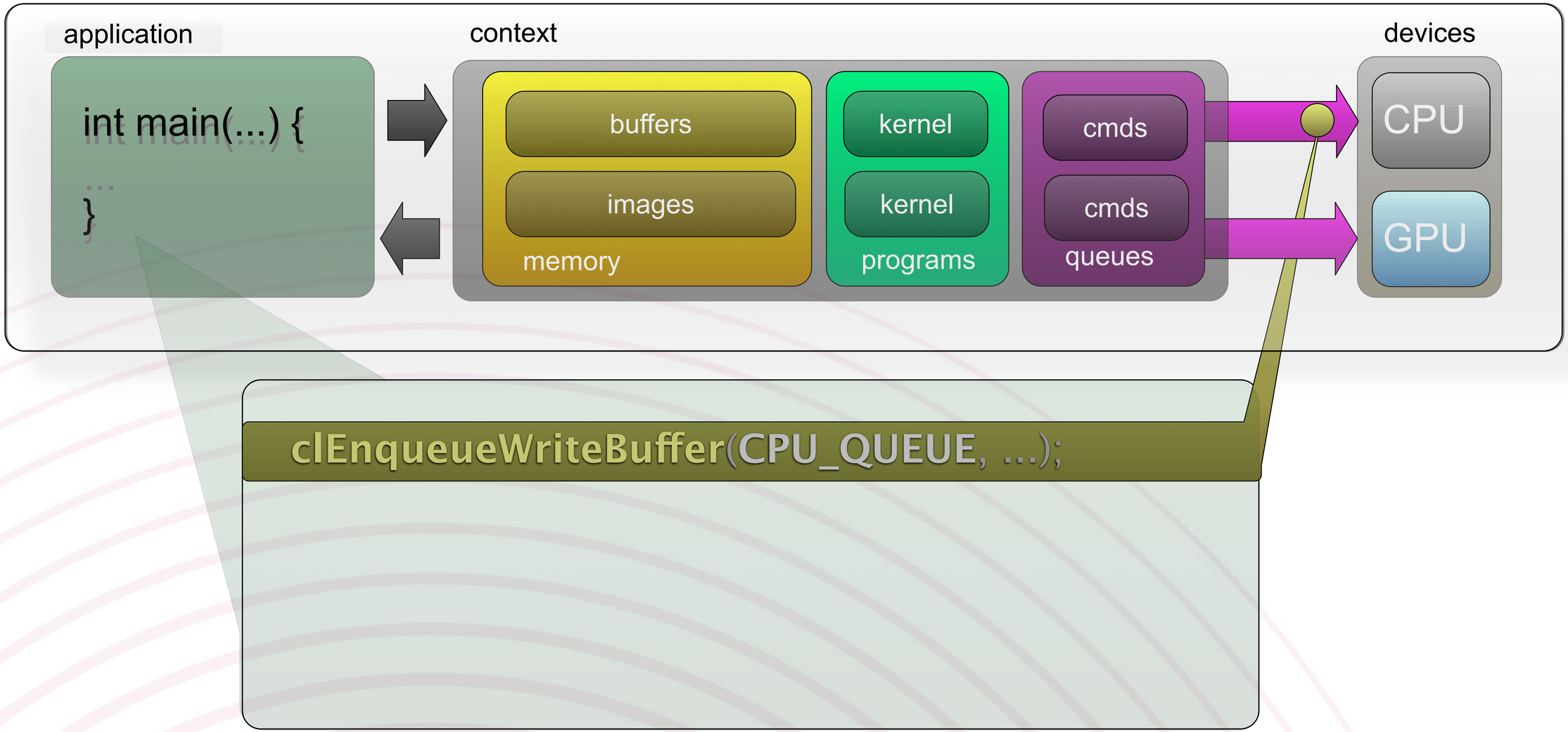
cl_command_queue **clCreateCommandQueue** (
cl_context *context*, cl_device_id *device*,
cl_command_queue_properties *properties*,
cl_int **errcode_ret*)

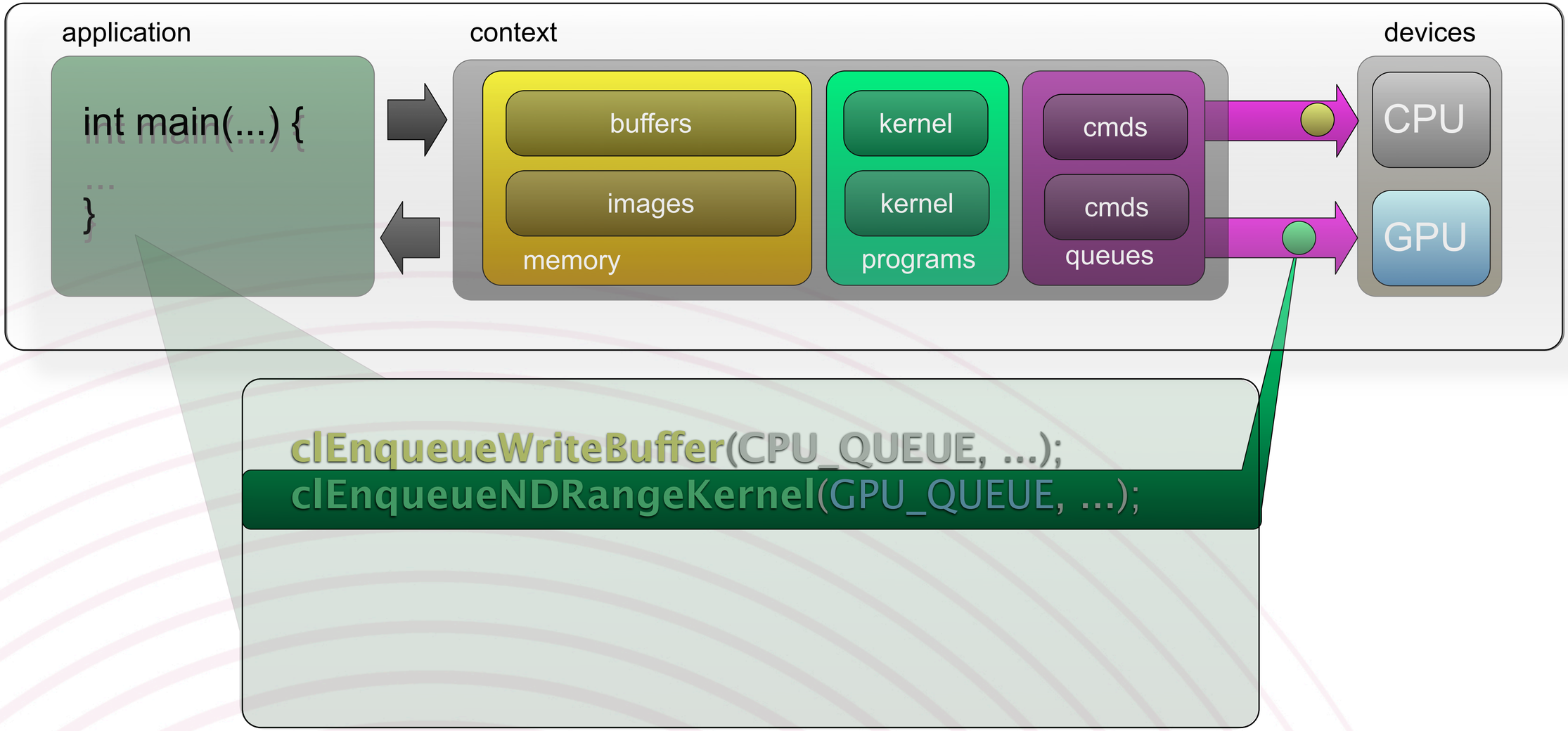
properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

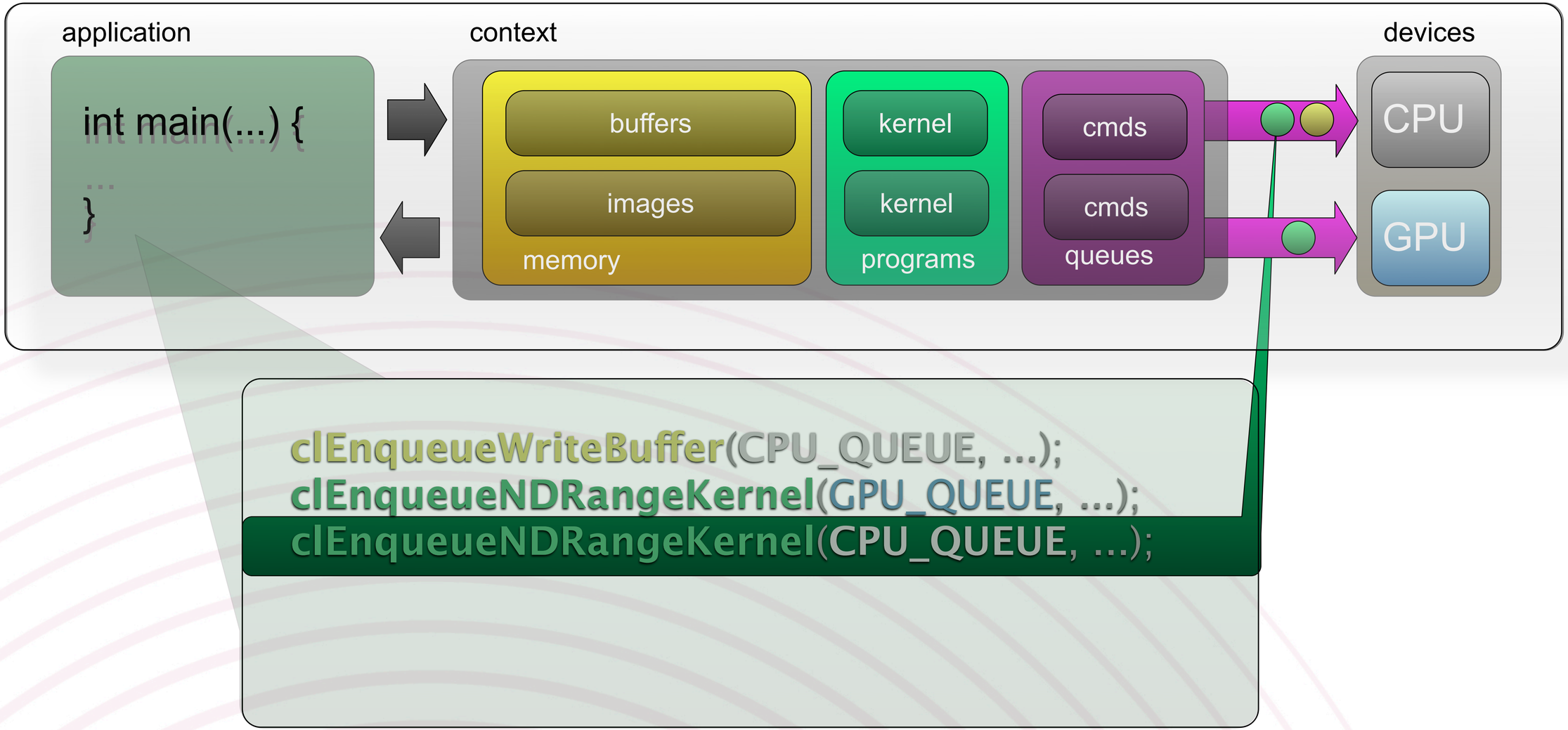
cl_int **clRetainCommandQueue** (
cl_command_queue *command_queue*)

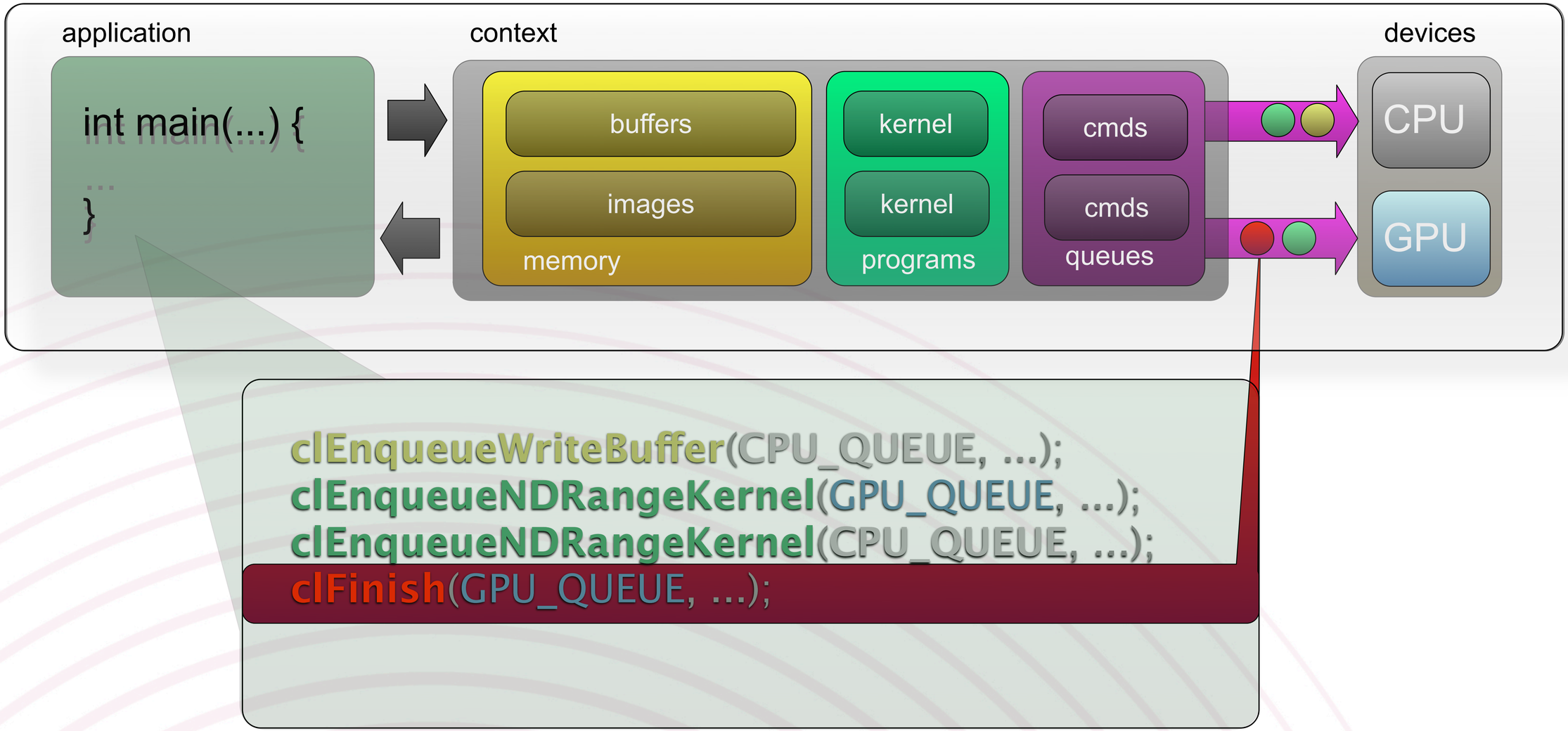
cl_int **clReleaseCommandQueue** (
cl_command_queue *command_queue*)

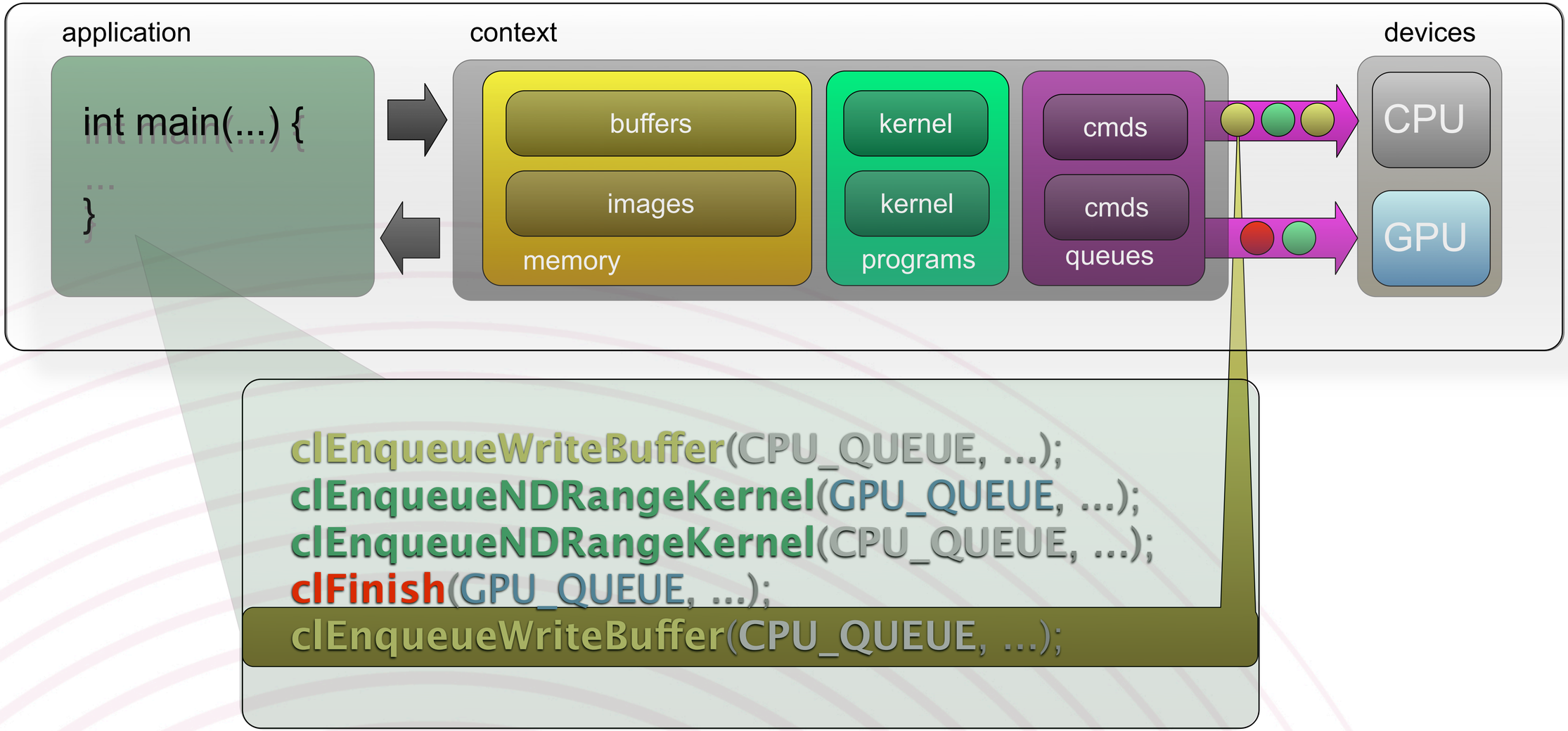
cl_int **clGetCommandQueueInfo** (
cl_command_queue *command_queue*,
cl_command_queue_info *param_name*,


application
int main(...) {
...
}
context
buffers
images
memory
kernel
kernel
programs
cmds
cmds
queues
devices
CPU
GPU
clEnqueueWriteBuffer(CPU_QUEUE, ...);


application
context
devices
int main(...) {
...
}
buffers
images
memory
kernel
kernel
programs
cmds
cmds
queues
CPU
GPU
clEnqueueWriteBuffer(CPU_QUEUE, ...);
clEnqueueNDRangeKernel(GPU_QUEUE, ...);


application
int main(...) {
...
}
context
buffers
images
memory
kernel
kernel
programs
cmds
cmds
queues
devices
CPU
GPU
clEnqueueWriteBuffer(CPU_QUEUE, ...);
clEnqueueNDRangeKernel(GPU_QUEUE, ...);
clEnqueueNDRangeKernel(CPU_QUEUE, ...);


application
context
devices
int main(...) {
...
}
buffers
images
memory
kernel
kernel
programs
cmds
cmds
queues
CPU
GPU
clEnqueueWriteBuffer(CPU_QUEUE, ...);
clEnqueueNDRangeKernel(GPU_QUEUE, ...);
clEnqueueNDRangeKernel(CPU_QUEUE, ...);
clFinish(GPU_QUEUE, ...);


application
int main(...) {
...
}
context
buffers
images
memory
kernel
kernel
programs
cmds
cmds
queues
devices
CPU
GPU
clEnqueueWriteBuffer(CPU_QUEUE, ...);
clEnqueueNDRangeKernel(GPU_QUEUE, ...);
clEnqueueNDRangeKernel(CPU_QUEUE, ...);
clFinish(GPU_QUEUE, ...);
clEnqueueWriteBuffer(CPU_QUEUE, ...);

Work Distribution

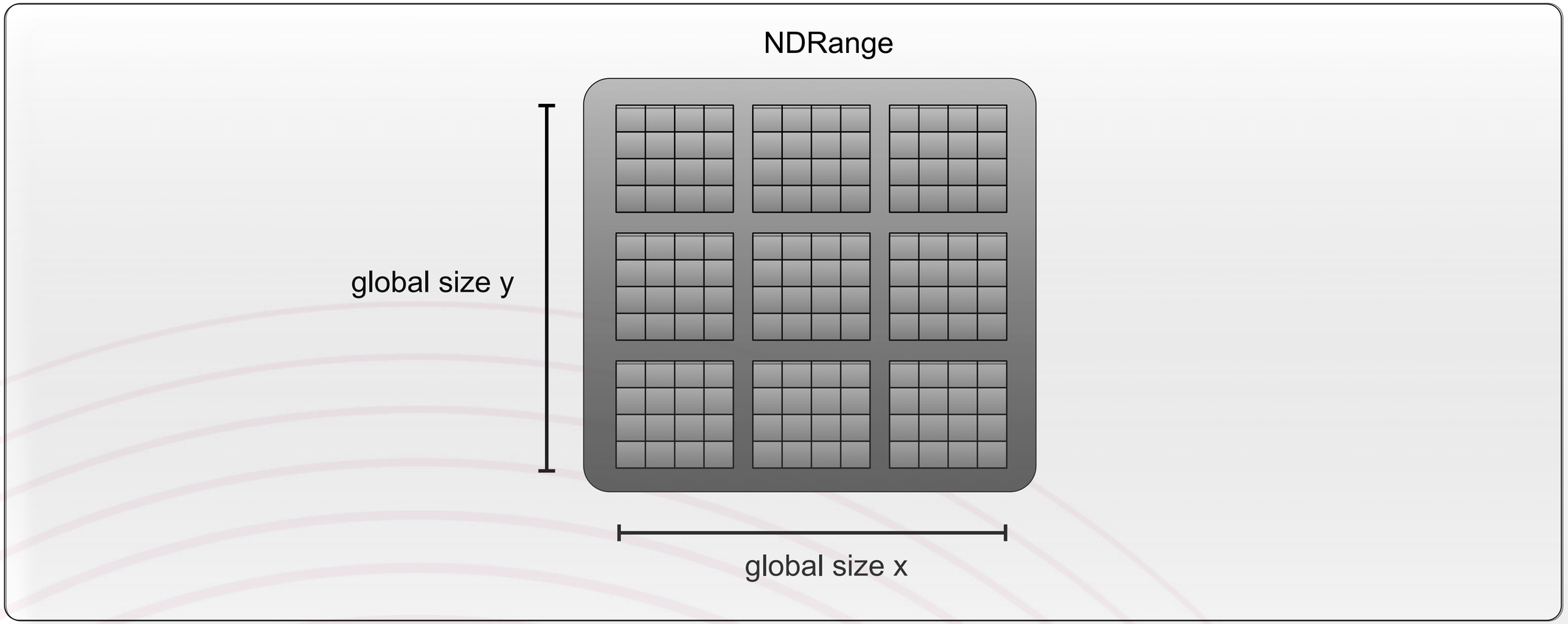
Logical groupings over a domain

- Kernels are executed across a domain of work-items
- Work-items are logically organised into work-groups
- Work-groups are executed in parallel

Work Distribution

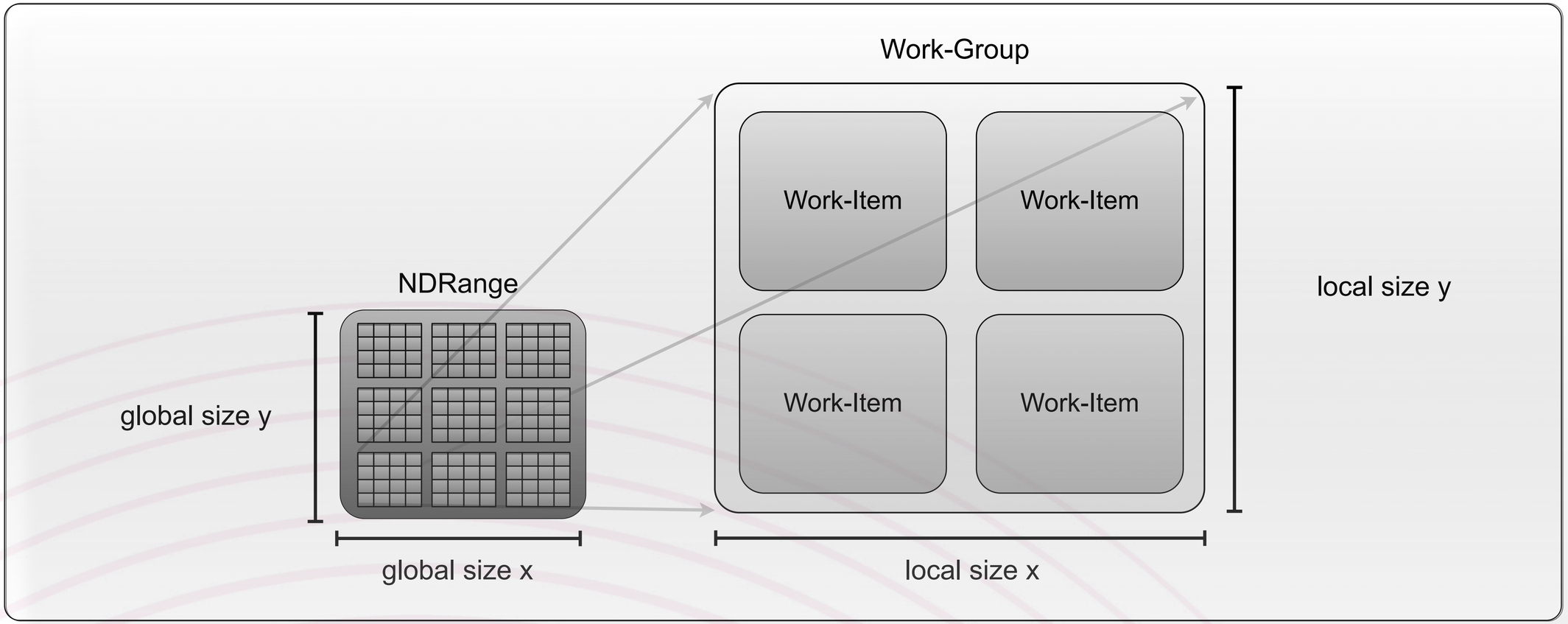
Logical groupings over a domain

- Global dimensions define range of computation
- Local dimensions define size of work-group
- Only work-items in a work-group can communicate



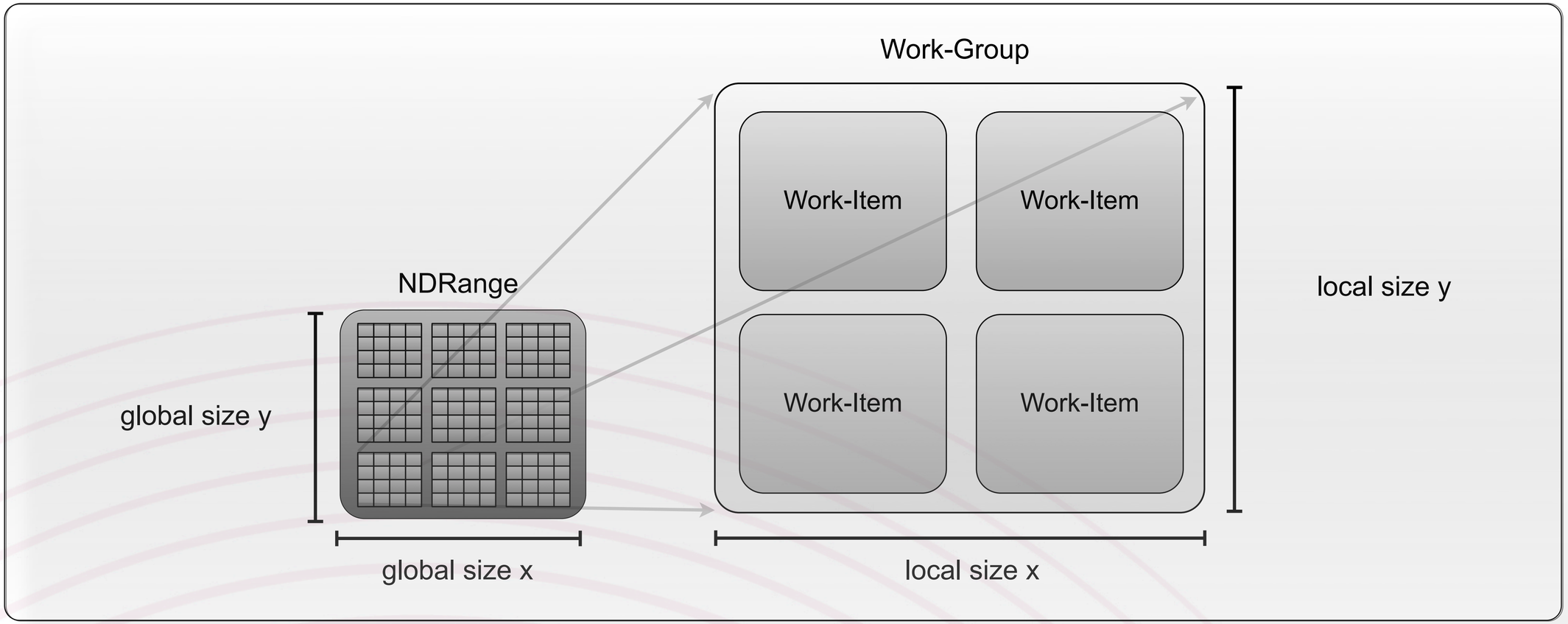
Host application invokes a kernel over an index space

Index space is an N-dimensional range (where N is 1, 2, or 3)



Global range is executed over local work-groups

Each work-group has a collection of work-items with a unique id



Work-items share resources in a work-group

Mitigates communication costs and allows group-wise synchronisation

```
__kernel void square(
    __global float* input, __global float* output)
{
    size_t i = get_global_id(0);
    output[i] = input[i] * input[i];
}
```

Built-in methods provide access to index space

Use the **get_global_id()** built-in for globally unique addresses

Use the **get_group_id()** for the logical group id spanning the ND-range

Use the **get_local_id()** for the local work-item address within a work-group

OpenCL API 1.1 Quick Reference Card - Page 3

__OPENCL_VERSION__	Substitutes integer version number.
__CL_VERSION_1_0__	Substitutes integer 100 for version 1.0
__CL_VERSION_1_1__	Substitutes integer 110 for version 1.1
__ENDIAN_LITTLE__	1 if device is little endian
__kernel_exec(X, type*n*)	Same as: __kernel __attribute__ ((work_group_size_hint(*X*, 1, 1))) __attribute__((vec_type_hint(*type*n*)))
__IMAGE_SUPPORT__	1 if images are supported
__FAST_RELAXED_MATH__	1 if –cl-fast-relaxed-math optimization option is specified

The values of the following symbolic constants are type float and are accurate within the precision of a single precision floating-point number.

MAXFLOAT	Value of max. non-infinite single-precision floating-point number.
HUGE_VALF	Positive float expression, evaluates to +infinity. Used as error value.
INFINITY	Constant float expression, positive or unsigned infinity.
NAN	Constant float expression, quiet NaN.
M_E_F	Value of e
M_LOG2E_F	Value of log2e
M_LOG10E_F	Value of log10e
M_LN2_F	Value of loge2
M_LN10_F	Value of loge10
M_PI_F	Value of π
M_PI_2_F	Value of π / 2
M_PI_4_F	Value of π / 4
M_1_PI_F	Value of 1 / π
M_2_PI_F	Value of 2 / π
M_2_SQRTPI_F	Value of 2 / √π
M_SQRT2_F	Value of √2
M_SQRT1_2_F	Value of 1 / √2

Work-Item Built-in Functions [6.11.1]

D is dimension index.

uint **get_work_dim** ()	Num. of dimensions in use
size_t **get_global_size** (uint *D*)	Num. of global work-items
size_t **get_global_id** (uint *D*)	Global work-item ID value
size_t **get_local_size** (uint *D*)	Num. of local work-items
size_t **get_local_id** (uint *D*)	Local work-item ID
size_t **get_num_groups** (uint *D*)	Num. of work-groups
size_t **get_group_id** (uint *D*)	Returns the work-group ID
size_t **get_global_offset** (uint *D*)	Returns global offset

intn, uintn, and ulongn must be scalar when *T* is scalar. *Q* is qualifier __global, __local, or __private. HN indicates that Half and Native variants are available by prepending "half_" or "native_" to function name. Prototypes shown in purple are half_ and native_ only. **Optional extensions enable double, doublen, half, and halfn types.**

T **acos** (*T*)	Arc cosine
T **acosh** (*T*)	Inverse hyperbolic cosine
T **acospi** (*T x*)	acos (*x*) / π
T **asin** (*T*)	Arc sine
T **asinh** (*T*)	Inverse hyperbolic sine
T **asinpi** (*T x*)	asin (*x*) / π
T **atan** (*T y_over_x*)	Arc tangent
T **atan2** (*T y*, *T x*)	Arc tangent of *y* / *x*
T **atanh** (*T*)	Hyperbolic arc tangent
T **atanpi** (*T x*)	atan (*x*) / π
T **atan2pi** (*T x*, *T y*)	atan2 (*x*, *y*) / π
T **cbrt** (*T*)	Cube root
T **ceil** (*T*)	Round to integer toward + infinity
T **copysign** (*T x*, *T y*)	*x* with sign changed to sign of *y*
T **cos** (*T*) HN	Cosine
T **cosh** (*T*)	Hyperbolic consine
T **cospi** (*T x*)	cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)	*x* / *y* (*T* may be float or float*n*)
T **erfc** (*T*)	Complementary error function
T **erf** (*T*)	Calculates error function of *T*
T **exp** (*T x*) HN	Exponential base e
T **exp2** (*T*) HN	Exponential base 2
T **exp10** (*T*) HN	Exponential base 10
T **fdim** (*T x*, *T y*)	"Positive difference" between *x* and *y*
T **floor** (*T*)	Round to integer toward - infinity
T **fma** (*T a*, *T b*, *T c*)	Multiply and add, then round
T **fmax** (*T x*, *T y*) halfn **fmax** (halfn *x*, half *y*) floatn **fmax**(floatn *x*, float *y*) doublen **fmax**(doublen *x*, double *y*)	Return *y* if *x* < *y*, otherwise it returns *x*
T **fmin** (*T x*, *T y*) halfn **fmin** (halfn *x*, half *y*) floatn **fmin**(floatn *x*, float *y*) doublen **fmin**(doublen *x*, double *y*)	Return *y* if *y* < *x*, otherwise it returns *x*
T **fmod** (*T x*, *T y*)	Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *Q T* **ptr*)	Fractional value in *x*
T **frexp** (*T x*, *Q* intn **exp*)	Extract mantissa and exponent
T **hypot** (*T x*, *T y*)	Square root of *x*^2+ *y*^2
intn **ilogb** (*T x*)	Return exponent as an integer value
T **ldexp** (*T x*, intn *n*) *T* **ldexp** (*T x*, int *n*)	*x* * 2^*n*
T **lgamma** (*T x*) *T* **lgamma_r** (*T x*, *Q* intn **signp*)	Log gamma function
T **log** (*T*) HN	Natural logarithm
T **log2** (*T*) HN	Base 2 logarithm
T **log10** (*T*) HN	Base 10 logarithm
T **log1p** (*T x*)	ln (1.0 + *x*)
T **logb** (*T x*)	Exponent of *x*
T **mad** (*T a*, *T b*, *T c*)	Approximates *a* * *b* + *c*
T **maxmag** (*T x*, *T y*)	Maximum magnitude of *x* and *y*
float **nan** (uintn *nancode*) floatn **nan** (uintn *nancode*) halfn **nan** (ushortn *nancode*) doublen **nan** (ulongn *nancode*)	Quiet NaN
T **nextafter** (*T x*, *T y*)	Next representable floating-point value following *x* in the direction of *y*
T **pow** (*T x*, *T y*)	Compute *x* to the power of *y* (*x*^*y*)
T **pown** (*T x*, intn *y*)	Compute *x*^*y*, where *y* is an integer
T **powr** (*T x*, *T y*) HN	Compute *x*^*y*, where *x* is >= 0
T **half_recip** (*T x*) *T* **native_recip** (*T x*)	1 / *x* (*T* may be float or float*n*)
T **remainder** (*T x*, *T y*)	Floating point remainder
T **remquo** (*T x*, *T y*, *Q* intn **quo*)	Floating point remainder and quotient
T **rint** (*T*)	Round to nearest even integer
T **rootn** (*T x*, intn *y*)	Compute *x* to the power of 1/*y*
T **round** (*T x*)	Integral value nearest to *x* rounding
T **rsqrt** (*T*) HN	Inverse square root
T **sin** (*T*) HN	Sine
T **sincos** (*T x*, *Q T* **cosval*)	Sine and cosine of *x*
T **sinh** (*T*)	Hyperbolic sine
T **sinpi** (*T x*)	sin (π *x*)
T **sqrt** (*T*) HN	Square root
T **tan** (*T*) HN	Tangent
T **tanh** (*T*)	Hyperbolic tangent
T **tanpi** (*T x*)	tan (π *x*)
T **tgamma** (*T*)	Gamma function
T **trunc** (*T*)	Round to integer toward zero

Memory Model

Hierarchy of address spaces

- Each address space is distinct and can't be intermixed
- All data movement between each address space must be done **explicity** (at all levels)

Memory Model

Uses relaxed consistency

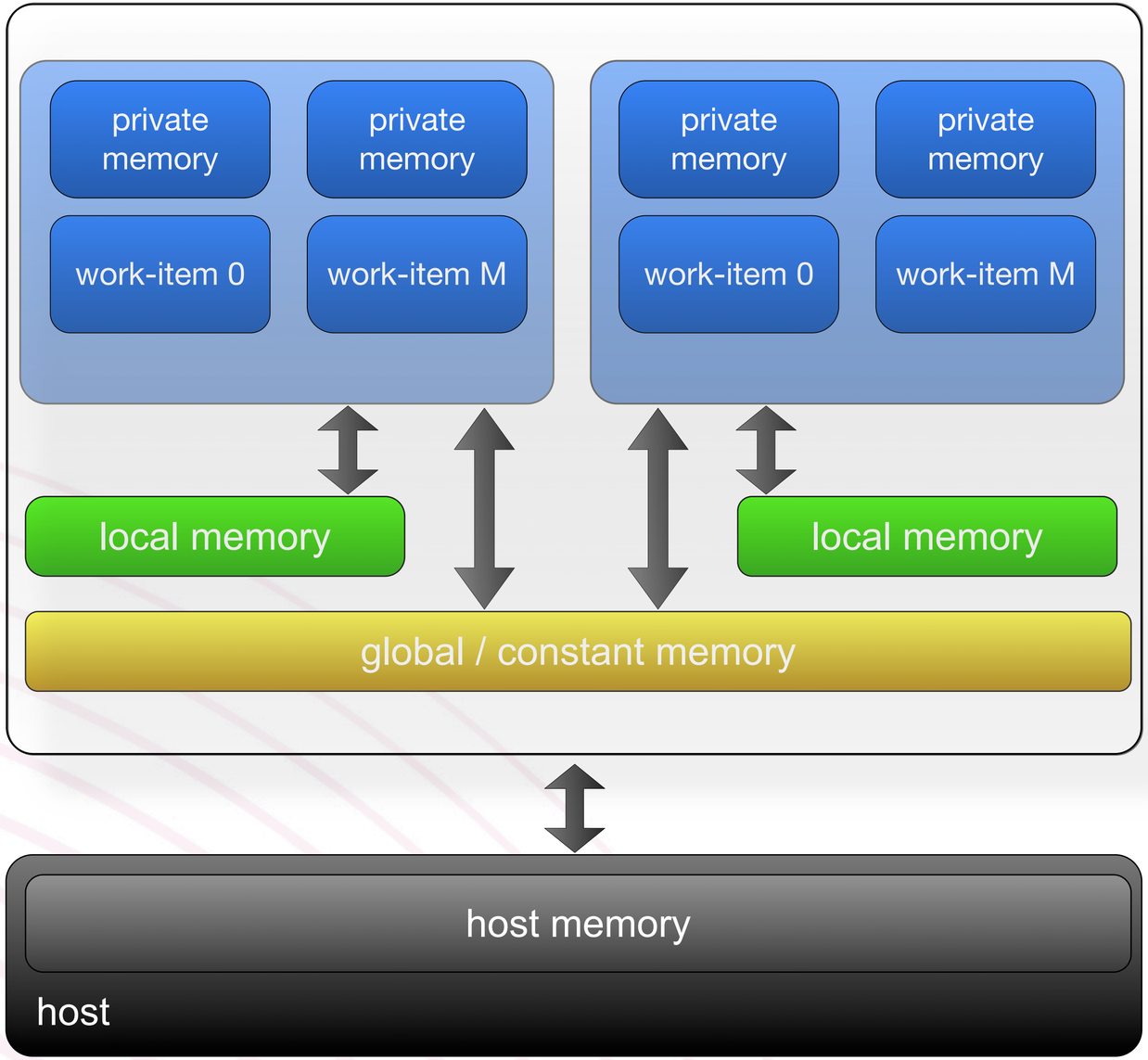
- State of memory visible to a work-item is not guaranteed to be consistent with all work-items
- If consistency is needed, synchronisation is **required**

Private:
Visible only to a single work-item

Local:
Shared within a work-group

Global/Constant::
Accessible by any work-item

Host:
Accessible only from application



float **dot** (float *p0*, float *p1*) float **dot** (float*n* *p0*, float*n* *p1*) double **dot** (double *p0*, double *p1*) double **dot** (double*n* *p0*, double*n* *p1*) half **dot** (half *p0*, half *p1*) half **dot** (half*n* *p0*, half*n* *p1*)	Dot product
float{3,4} **cross** (float{3,4} *p0*, float{3,4} *p1*) double{3,4} **cross** (double{3,4} *p0*, double{3,4} *p1*) half{3,4} **cross** (half{3,4} *p0*, half{3,4} *p1*)	Cross product

double **distance** (double*n* *p0*, double*n* *p1*) half **distance** (half *p0*, half *p1*) half **distance** (half*n* *p0*, half*n* *p1*)	
float **length** (float *p*) float **length** (float*n* *p*) double **length** (double *p*) double **length** (double*n* *p*) half **length** (half *p*) half **length** (half*n* *p*)	Vector l

Relational Built-in Functions [6.11.6]

T is type float, float*n*, char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n* (and optionally double, double*n*). *S* is type char, char*n*, short, short*n*, int, int*n*, long, or long*n*. *U* is type uchar, uchar*n*, ushort, ushort*n*, uint, uint*n*, ulong, or ulong*n*. **Optional extensions enable double, double*n*, and half*n* types.**

int **isequal** (float *x*, float *y*) int*n* **isequal** (float*n* *x*, float*n* *y*) int **isequal** (double *x*, double *y*) long*n* **isequal** (double*n* *x*, double*n* *y*) int **isequal** (half *x*, half *y*) short*n* **isequal** (half*n* *x*, half*n* *y*)	Compare of *x* == *y*
int **isnotequal** (float *x*, float *y*) int*n* **isnotequal** (float*n* *x*, float*n* *y*) int **isnotequal** (double *x*, double *y*) long*n* **isnotequal** (double*n* *x*, double*n* *y*) int **isnotequal** (half *x*, half *y*) short*n* **isnotequal** (half*n* *x*, half*n* *y*)	Compare of *x* != *y*
int **isgreater** (float *x*, float *y*) int*n* **isgreater** (float*n* *x*, float*n* *y*) int **isgreater** (double *x*, double *y*) long*n* **isgreater** (double*n* *x*, double*n* *y*) int **isgreater** (half *x*, half *y*) short*n* **isgreater** (half*n* *x*, half*n* *y*)	Compare of *x* > *y*
int **isgreaterequal** (float *x*, float *y*) int*n* **isgreaterequal** (float*n* *x*, float*n* *y*) int **isgreaterequal** (double *x*, double *y*) long*n* **isgreaterequal** (double*n* *x*, double*n* *y*) int **isgreaterequal** (half *x*, half *y*) short*n* **isgreaterequal** (half*n* *x*, half*n* *y*)	Compare of *x* >= *y*
int **isless** (float *x*, float *y*) int*n* **isless** (float*n* *x*, float*n* *y*) int **isless** (double *x*, double *y*) long*n* **isless** (double*n* *x*, double*n* *y*) int **isless** (half *x*, half *y*) short*n* **isless** (half*n* *x*, half*n* *y*)	Compare of *x* < *y*
int **islessequal** (float *x*, float *y*) int*n* **islessequal** (float*n* *x*, float*n* *y*) int **islessequal** (double *x*, double *y*) long*n* **islessequal** (double*n* *x*, double*n* *y*) int **islessequal** (half *x*, half *y*) short*n* **islessequal** (half*n* *x*, half*n* *y*)	Compare of *x* <= *y*
int **islessgreater** (float *x*, float *y*) int*n* **islessgreater** (float*n* *x*, float*n* *y*) int **islessgreater** (double *x*, double *y*) long*n* **islessgreater** (double*n* *x*, double*n* *y*) int **islessgreater** (half *x*, half *y*) short*n* **islessgreater** (half*n* *x*, half*n* *y*)	Compare of (*x* < *y*) || (*x* > *y*)
int **isfinite** (float) int*n* **isfinite** (float*n*) int **isfinite** (double) long*n* **isfinite** (double*n*) int **isfinite** (half) short*n* **isfinite** (half*n*)	Test for finite value

int **isinf** (float) int*n* **isinf** (float*n*) int **isinf** (double) long*n* **isinf** (double*n*) int **isinf** (half) short*n* **isinf** (half*n*)	Test for +ve or –ve infinity
int **isnan** (float) int*n* **isnan** (float*n*) int **isnan** (double) long*n* **isnan** (double*n*) int **isnan** (half) short*n* **isnan** (half*n*)	Test for a NaN
int **isnormal** (float) int*n* **isnormal** (float*n*) int **isnormal** (double) long*n* **isnormal** (double*n*) int **isnormal** (half) short*n* **isnormal** (half*n*)	Test for a nor value
int **isordered** (float *x*, float *y*) int*n* **isordered** (float*n* *x*, float*n* *y*) int **isordered** (double *x*, double *y*) long*n* **isordered** (double*n* *x*, double*n* *y*) int **isordered** (half *x*, half *y*) short*n* **isordered** (half*n* *x*, half*n* *y*)	Test if argume are ordere
int **isunordered** (float *x*, float *y*) int*n* **isunordered** (float*n* *x*, float*n* *y*) int **isunordered** (double *x*, double *y*) long*n* **isunordered** (double*n* *x*, double*n* *y*) int **isunordered** (half *x*, half *y*) short*n* **isunordered** (half*n* *x*, half*n* *y*)	Test if argum are unord
int **signbit** (float) int*n* **signbit** (float*n*) int **signbit** (double) long*n* **signbit** (double*n*) int **signbit** (half) short*n* **signbit** (half*n*)	Test for sign
int **any** (*S x*)	1 if MSB in ar component o set; else 0
int **all** (*S x*)	1 if MSB in all components set; else 0
T **bitselect** (*T a*, *T b*, *T c*) half*n* **bitselect** (half*n* *a*, half*n* *b*, half*n* *c*) double*n* **bitselect** (double*n* *a*, double*n* *b*, double*n* *c*)	Each bit of re correspondin *a* if correspo of *c* is 0
T **select** (*T a*, *T b*, *S c*) *T* **select** (*T a*, *T b*, *U c*) double*n* **select** (double*n*, double*n*, long*n*) double*n* **select** (double*n*, double*n*, ulong*n*) half*n* **select** (half*n*, half*n*, short*n*) half*n* **select** (half*n*, half*n*, ushort*n*)	For each com of a vector ty result[*i*] = if c[*i*] is set ? *b* For scalar typ result = *c* ? *b*

Atomic Functions [6.11.11, 9.4]

T is type int or unsigned int. *T* may also be type float for **atomic_xchg**, and type long or ulong for extended 64-bit atomic functions. *Q* is volatile __global or volatile __local, except *Q* must be volatile __global for **atomic_xchg** when *T* is float.

The built-in atomic functions for 32-bit values begin with atomic_ while the extended 64-bit atomic functions begin with atom_. For example:

Built-in atomic function	Extended atomic function
atomic_add ()	**atom_add** ()

Extended 64-bit atomic functions are enabled by the following pragma; *extension-name* is one of cl_khr_int64_{base, extended}_atomics:

#pragma OPENCL EXTENSION *extension-name* : enable

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

Async Copies and Prefetch Functions [6.11.10]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, ulong*n*, float, float*n*, and optionally half*n* double, double*n*. **Optional extensions enable the half*n*, double, and double*n* types.**

event_t **async_work_group_copy** (__local *T* *dst, const __global *T* *src, size_t *num_gentypes*, event_t *event*) event_t **async_work_group_copy** (__global *T* *dst, const __local *T* *src, size_t *num_gentypes*, event_t *event*)	Copies *num_gentypes T* elements from *src* to *dst*
event_t **async_work_group_strided_copy** (__local *T* *dst, const __global *T* *src, size_t *num_gentypes*, size_t *src_stride*, event_t *event*) event_t **async_work_group_strided_copy** (__global *T* *dst, const __local	Copies *num_gentypes T* elements from *src* to *dst*

OpenCL API 1.1 Quick Reference Card - Page 4

Atomic Functions [6.11.11, 9.4]

T is type int or unsigned int. ***T*** may also be type float for **atomic_xchg**, and type long or ulong for extended 64-bit atomic functions. ***Q*** is volatile __global or volatile __local, except ***Q*** must be volatile __global for **atomic_xchg** when ***T*** is float.

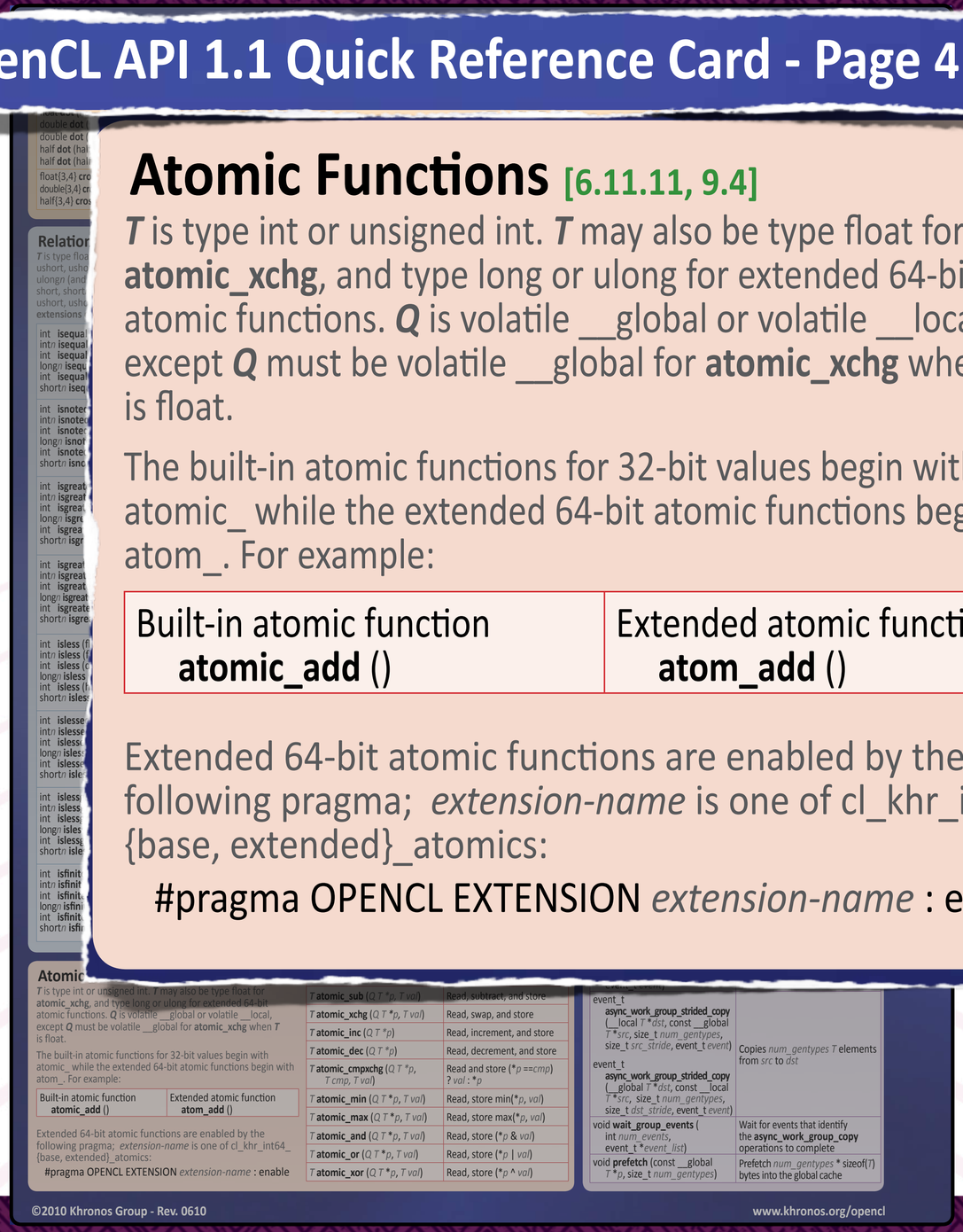
The built-in atomic functions for 32-bit values begin with atomic_ while the extended 64-bit atomic functions begin with atom_. For example:

Built-in atomic function	Extended atomic function
atomic_add ()	**atom_add** ()

Extended 64-bit atomic functions are enabled by the following pragma; *extension-name* is one of cl_khr_int64_{base, extended}_atomics:

```
#pragma OPENCL EXTENSION extension-name : enable
```

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



double **dot** (double*n* *p0*, double*n* *p1*)	
double **dot** (double*n* *p0*, double*n* *p1*)	
half **dot** (half*n* *p0*, half*n* *p1*)	
half **dot** (half*n* *p0*, half*n* *p1*)	
float{3,4} **cross** (float{3,4} *p0*, float{3,4} *p1*) double{3,4} **cross** (double{3,4} *p0*, double{3,4} *p1*) half{3,4} **cross** (half{3,4} *p0*, half{3,4} *p1*)	Cross product

float **length** (float *p*)	
float **length** (float*n* *p*)	
double **length** (double *p*)	
double **length** (double*n* *p*)	
half **length** (half *p*)	
half **length** (half*n* *p*)	

Relational Built-in Functions [6.11.6]

T is type float, float*n*, char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n* (and optionally double, double*n*). *S* is type char, char*n*, short, short*n*, int, int*n*, long, or long*n*. *U* is type uchar, uchar*n*, ushort, ushort*n*, uint, uint*n*, ulong, or ulong*n*. **Optional extensions enable double, double*n*, and half*n* types.**

int **isequal** (float *x*, float *y*) int*n* **isequal** (float*n* *x*, float*n* *y*) int **isequal** (double *x*, double *y*) long*n* **isequal** (double*n* *x*, double*n* *y*) int **isequal** (half *x*, half *y*) short*n* **isequal** (half*n* *x*, half*n* *y*)	Compare of *x* == *y*
int **isnotequal** (float *x*, float *y*) int*n* **isnotequal** (float*n* *x*, float*n* *y*) int **isnotequal** (double *x*, double *y*) long*n* **isnotequal** (double*n* *x*, double*n* *y*) int **isnotequal** (half *x*, half *y*) short*n* **isnotequal** (half*n* *x*, half*n* *y*)	Compare of *x* != *y*
int **isgreater** (float *x*, float *y*) int*n* **isgreater** (float*n* *x*, float*n* *y*) int **isgreater** (double *x*, double *y*) long*n* **isgreater** (double*n* *x*, double*n* *y*) int **isgreater** (half *x*, half *y*) short*n* **isgreater** (half*n* *x*, half*n* *y*)	Compare of *x* > *y*
int **isgreaterequal** (float *x*, float *y*) int*n* **isgreaterequal** (float*n* *x*, float*n* *y*) int **isgreaterequal** (double *x*, double *y*) long*n* **isgreaterequal** (double*n* *x*, double*n* *y*) int **isgreaterequal** (half *x*, half *y*) short*n* **isgreaterequal** (half*n* *x*, half*n* *y*)	Compare of *x* >= *y*
int **isless** (float *x*, float *y*) int*n* **isless** (float*n* *x*, float*n* *y*) int **isless** (double *x*, double *y*) long*n* **isless** (double*n* *x*, double*n* *y*) int **isless** (half *x*, half *y*) short*n* **isless** (half*n* *x*, half*n* *y*)	Compare of *x* < *y*
int **islessequal** (float *x*, float *y*) int*n* **islessequal** (float*n* *x*, float*n* *y*) int **islessequal** (double *x*, double *y*) long*n* **islessequal** (double*n* *x*, double*n* *y*) int **islessequal** (half *x*, half *y*) short*n* **islessequal** (half*n* *x*, half*n* *y*)	Compare of *x* <= *y*
int **isgreater** (float *x*, float *y*) int*n* **islessgreater** (float*n* *x*, float*n* *y*) int **islessgreater** (double *x*, double *y*) long*n* **islessgreater** (double*n* *x*, double*n* *y*) int **islessgreater** (half *x*, half *y*) short*n* **islessgreater** (half*n* *x*, half*n* *y*)	Compare of (*x* < *y*) || (*x* > *y*)
int **isfinite** (float) int*n* **isfinite** (float*n*) int **isfinite** (double) long*n* **isfinite** (double*n*) int **isfinite** (half) short*n* **isfinite** (half*n*)	Test for finite value

int **isinf** (float) int*n* **isinf** (float*n*) int **isinf** (double) long*n* **isinf** (double*n*) int **isinf** (half) short*n* **isinf** (half*n*)	
int **isnan** (float) int*n* **isnan** (float*n*) int **isnan** (double) long*n* **isnan** (double*n*) int **isnan** (half) short*n* **isnan** (half*n*)	
int **isnormal** (float) int*n* **isnormal** (float*n*) int **isnormal** (double) long*n* **isnormal** (double*n*) int **isnormal** (half) short*n* **isnormal** (half*n*)	
int **isordered** (float *x*, float *y*) int*n* **isordered** (float*n* *x*, float*n* *y*) int **isordered** (double *x*, double *y*) long*n* **isordered** (double*n* *x*, double*n* int **isordered** (half *x*, half *y*) short*n* **isordered** (half*n* *x*, half*n* *y*)	
int **isunordered** (float *x*, float *y*) int*n* **isunordered** (float*n* *x*, float*n* *y*) int **isunordered** (double *x*, double long*n* **isunordered** (double*n* *x*, doubl int **isunordered** (half *x*, half *y*) short*n* **isunordered** (half*n* *x*, half*n* *y*	
int **signbit** (float) int*n* **signbit** (float*n*) int **signbit** (double) long*n* **signbit** (double*n*) int **signbit** (half) short*n* **signbit** (half*n*)	
int **any** (*S x*)	
int **all** (*S x*)	
T **bitselect** (*T a*, *T b*, *T c*) half*n* **bitselect** (half*n* *a*, half*n* *b*, half double*n* **bitselect** (double*n* *a*, double*n* *b*, double*n* *c*)	
T **select** (*T a*, *T b*, *S c*) *T* **select** (*T a*, *T b*, *U c*) double*n* **select** (double*n*, double*n*, l double*n* **select** (double*n*, double*n*, ul half*n* **select** (half*n*, half*n*, short*n*) half*n* **select** (half*n*, half*n*, ushort*n*)	

Atomic Functions [6.11.11, 9.4]

T is type int or unsigned int. *T* may also be type float for **atomic_xchg**, and type long or ulong for extended 64-bit atomic functions. *Q* is volatile __global or volatile __local, except *Q* must be volatile __global for **atomic_xchg** when *T* is float.

The built-in atomic functions for 32-bit values begin with atomic_ while the extended 64-bit atomic functions begin with atom_. For example:

Built-in atomic function	Extended atomic function
atomic_add ()	**atom_add** ()

Extended 64-bit atomic functions are enabled by the following pragma; *extension-name* is one of cl_khr_int64_{base, extended}_atomics:

#pragma OPENCL EXTENSION *extension-name* : enable

T **atomic_add** (*Q T *p*, *T val*)	Re
T **atomic_sub** (*Q T *p*, *T val*)	Re
T **atomic_xchg** (*Q T *p*, *T val*)	Re
T **atomic_inc** (*Q T *p*)	Re
T **atomic_dec** (*Q T *p*)	Re
T **atomic_cmpxchg** (*Q T *p*, *T cmp*, *T val*)	Re ?
T **atomic_min** (*Q T *p*, *T val*)	Re
T **atomic_max** (*Q T *p*, *T val*)	Re
T **atomic_and** (*Q T *p*, *T val*)	Re
T **atomic_or** (*Q T *p*, *T val*)	Re
T **atomic_xor** (*Q T *p*, *T val*)	Re

Vector Data Load/Store Functions [6.11.7]

Q is an Address Space Qualifier listed in **6.5** unless otherwise noted. ***R*** defaults to the current rounding mode, or is one of the Rounding Modes listed in **6.2.3.2**. ***T*** is type char, uchar, short, ushort, int, uint, long, ulong, half, or float (or optionally double). ***Tn*** refers to the vector form of type *T*. **Optional extensions enable the double, double*n*, half, and half*n* types.**

Tn **vload*n*** (size_t *offset*, const *Q T *p*)	Read vector data from memory
void **vstore*n*** (*Tn data*, size_t *offset*, *Q T *p*)	Write vector data to memory *(Q in this function cannot be __constant)*
float **vload_half** (size_t *offset*, const *Q* half *p*)	Read a half from memory
float*n* **vload_half*n*** (size_t *offset*, const *Q* half *p*)	Read multiple halfs from memory
void **vstore_half** (float *data*, size_t *offset*, *Q* half *p*) void **vstore_half_***R* (float *data*,	Write a half to memory

__LINE__	Integer line number
__OPENCL_VERSION__	Integer version number
__CL_VERSION_1_0__	Substitutes integer 100 for version 1.0
__CL_VERSION_1_1__	Substitutes integer 110 for version 1.1
__ENDIAN_LITTLE__	1 if device is little endian
__kernel_exec(X, type*n*)	Same as: __kernel __attribute__((work_group_size_hint(*X*, 1, 1))) __attribute__((vec_type_hint(*type*n*)))
__IMAGE_SUPPORT__	1 if images are supported
__FAST_RELAXED_MATH__	1 if -cl-fast-relaxed-math optimization option is specified

Math Constants

The values of the follow constants are type float accurate within the pre precision floating-point

MAXFLOAT	Value non- preci point
HUGE_VALF	Positi expre to +in error

Work-Item Built-in Functions [6.11.1]

D is dimension index.

uint **get_work_dim** ()	Num. of dimensions in use	size_t **get_local_id** (uint *D*)
size_t **get_global_size** (uint *D*)	Num. of global work-items	size_t **get_num_groups** (uint *D*)
size_t **get_global_id** (uint *D*)	Global work-item ID value	size_t **get_group_id** (uint *D*)
size_t **get_local_size** (uint *D*)	Num. of local work-items	size_t **get_global_offset** (uint *D*)

Integer Built-in Functions [6.11.3]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n*. *U* is the unsigned version of *T*. *S* is the scalar version of *T*.

U **abs** (*T x*)	| *x* |
U **abs_diff** (*T x*, *T y*)	| *x* – *y* | without modulo overflow
T **add_sat** (*T x*, *T y*)	*x* + *y* and saturates the result
T **hadd** (*T x*, *T y*)	(*x* + *y*) >> 1 without mod. overflow
T **rhadd** (*T x*, *T y*)	(*x* + *y* + 1) >> 1
T **clz** (*T x*)	Number of leading 0-bits in *x*
T **clamp** (*T x*, *T min*, *T max*) *T* **clamp** (*T x*, *S min*, *S max*)	min(max(*x*, *minval*), *maxval*)
T **mad_hi** (*T a*, *T b*, *T c*)	mul_hi(*a*, *b*) + *c*
T **mad_sat** (*T a*, *T b*, *T c*)	*a* * *b* + *c* and saturates the result
T **max** (*T x*, *T y*) *T* **max** (*T x*, *S y*)	*y* if *x* < *y*, otherwise it returns *x*
T **min** (*T x*, *T y*) *T* **min** (*T x*, *S y*)	*y* if *y* < *x*, otherwise it returns *x*
T **mul_hi** (*T x*, *T y*)	high half of the product of *x* and *y*
T **rotate** (*T v*, *T i*)	result[*indx*] = *v*[*indx*] << *i*[*indx*]
T **sub_sat** (*T x*, *T y*)	*x* - *y*

For upsample, scalar types are perm

short*n* **upsample** (char*n hi*, uchar*n lo*)	resu
ushort*n* **upsample** (uchar*n hi*, uchar*n lo*)	resu
int*n* **upsample** (short*n hi*, ushort*n lo*)	resu
uint*n* **upsample** (ushort*n hi*, ushort*n lo*)	resu
long*n* **upsample** (int*n hi*, uint*n lo*)	resu
ulong*n* **upsample** (uint*n hi*, uint*n lo*)	resu

The following fast integer functi of kernels. In these functions, T int8, int16, uint, uint2, uint4, ui

T **mad24** (*T a*, *T b*, *T c*)	Mul 32-b
T **mul24** (*T a*, *T b*)	Mul

Math Built-in Functions [6.11.2]

T is type float or float*n* (or optionally double, double*n*, or half*n*). int*n*, uint*n*, and ulong*n* must be scalar when *T* is scalar. *Q* is qualifier __global, __local, or __private. **HN** indicates that Half and Native variants are available by prepending "half_" or "native_" to function name. Prototypes shown in purple are half_ and native_ only. **Optional extensions enable double, double*n*, half, and half*n* types.**

T **acos** (*T*)		Arc cosine
T **acosh** (*T*)		Inverse hyperbolic cosine
T **acospi** (*T x*)		acos (*x*) / π
T **asin** (*T*)		Arc sine
T **asinh** (*T*)		Inverse hyperbolic sine
T **asinpi** (*T x*)		asin (*x*) / π
T **atan** (*T y_over_x*)		Arc tangent
T **atan2** (*T y*, *T x*)		Arc tangent of *y* / *x*
T **atanh** (*T*)		Hyperbolic arc tangent
T **atanpi** (*T x*)		atan (*x*) / π
T **atan2pi** (*T x*, *T y*)		atan2 (*x*, *y*) / π
T **cbrt** (*T*)		Cube root
T **ceil** (*T*)		Round to integer toward + infinity
T **copysign** (*T x*, *T y*)		*x* with sign changed to sign of *y*
T **cos** (*T*)	HN	Cosine
T **cosh** (*T*)		Hyperbolic consine
T **cospi** (*T x*)		cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)		*x* / *y* (*T* may be float or float*n*)
T **erfc** (*T*)		Complementary error function
T **erf** (*T*)		Calculates error function of *T*
T **exp** (*T x*)	HN	Exponential base e
T **exp2** (*T*)	HN	Exponential base 2
T **exp10** (*T*)	HN	Exponential base 10
T **expm1** (*T x*)		e
T **fabs** (*T*)		A
T **fdim** (*T x*, *T y*)		
T **floor** (*T*)		F
T **fma** (*T a*, *T b*, *T c*)		M
T **fmax** (*T x*, *T y*) half*n* **fmax** (half*n x*, half *y*) float*n* **fmax**(float*n x*, float *y*) double*n* **fmax**(double*n x*, double *y*)		
T **fmin** (*T x*, *T y*) half*n* **fmin** (half*n x*, half *y*) float*n* **fmin**(float*n x*, float *y*) double*n* **fmin**(double*n x*, double *y*)		
T **fmod** (*T x*, *T y*)		
T **fract** (*T x*, *Q T *iptr*)		
T **frexp** (*T x*, *Q* int*n* **exp*)		
T **hypot** (*T x*, *T y*)		
int*n* **ilogb** (*T x*)		
T **ldexp** (*T x*, int*n n*) *T* **ldexp** (*T x*, int *n*)		
T **lgamma** (*T x*) *T* **lgamma_r** (*T x*, *Q* int*n* **signp*)		
T **log** (*T*)	HN	
T **log2** (*T*)	HN	
T **log10** (*T*)	HN	
T **log1p** (*T x*)		
T **logb** (*T x*)		
T **mad** (*T a*, *T b*, *T c*)		
T **maxmag** (*T x*, *T y*)		

Common Built-in Functions [6.11.4]

T is type float or float*n* (or optionally double, double*n*, or half*n*).
Optional extensions enable double, double*n*, and half*n* types.

T **clamp** (*T x*, *T min*, *T max*) float*n* **clamp** (float*n x*, float *min*, float *max*) double*n* **clamp** (double*n x*, double *min*, double *max*) half*n* **clamp** (half*n x*, half *min*, half *max*)	Clamp *x* to range given by *min*, *max*
T **degrees** (*T radians*)	*radians* to degrees
T **max** (*T x*, *T y*) float*n* **max** (float*n x*, float *y*) double*n* **max** (double*n x*, double *y*) half*n* **max** (half*n x*, half *y*)	Max of *x* and *y*
T **min** (*T x*, *T y*) float*n* **min** (float*n x*, float *y*) double*n* **min** (double*n x*, double *y*) half*n* **min** (half*n x*, half *y*)	Min of *x* and *y*
T **mix** (*T x*, *T y*, *T a*) float*n* **mix** (float*n x*, float *y*, float *a*) double*n* **mix** (double*n x*, double *y*, double *a*)	Linear blend of *x* and *y*

OpenCL API 1.1 Quick Reference Card - Page 3

Integer Built-in Functions [6.11.3]

T is type char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n*. ***U*** is the unsigned version of ***T***. ***S*** is the scalar version of ***T***.

U **abs** (*T x*)	| *x* |
U **abs_diff** (*T x*, *T y*)	| *x* – *y* | without modulo overflow
T **add_sat** (*T x*, *T y*)	*x* + *y* and saturates the result
T **hadd** (*T x*, *T y*)	(*x* + *y*) >> 1 without mod. overflow
T **rhadd** (*T x*, *T y*)	(*x* + *y* + 1) >> 1
T **clz** (*T x*)	Number of leading 0-bits in *x*
T **clamp** (*T x*, *T min*, *T max*) *T* **clamp** (*T x*, *S min*, *S max*)	min(max(*x*, *minval*), *maxval*)
T **mad_hi** (*T a*, *T b*, *T c*)	mul_hi(*a*, *b*) + *c*
T **mad_sat** (*T a*, *T b*, *T c*)	*a* * *b* + *c* and saturates the result
T **max** (*T x*, *T y*) *T* **max** (*T x*, *S y*)	*y* if *x* < *y*, otherwise it returns *x*
T **min** (*T x*, *T y*)	*y* if *y* < *x*, otherwise it returns *x*
T **min** (*T x*, *S y*)	*y* if *y* < *x*, otherwise it returns *x*
T **mul_hi** (*T x*, *T y*)	high half of the product of *x* and *y*
T **rotate** (*T v*, *T i*)	result[*indx*] = *v*[*indx*] << *i*[*indx*]
T **sub_sat** (*T x*, *T y*)	*x* - *y* and saturates the result
For ***upsample****, scalar types are permitted for the vector types below.*	
short*n* **upsample** (char*n* *hi*, uchar*n* *lo*)	result[*i*]= ((short)*hi*[*i*]<< 8) | *lo*[*i*]
ushort*n* **upsample** (uchar*n* *hi*, uchar*n* *lo*)	result[*i*]=((ushort)*hi*[*i*]<< 8) | *lo*[*i*]
int*n* **upsample** (short*n* *hi*, ushort*n* *lo*)	result[*i*]=((int)*hi*[*i*]<< 16) | *lo*[*i*]
uint*n* **upsample** (ushort*n* *hi*, ushort*n* *lo*)	result[*i*]=((uint)*hi*[*i*]<< 16) | *lo*[*i*]
long*n* **upsample** (int*n* *hi*, uint*n* *lo*)	result[*i*]=((long)*hi*[*i*]<< 32) | *lo*[*i*]
ulong*n* **upsample** (uint*n* *hi*, uint*n* *lo*)	result[*i*]=((ulong)*hi*[*i*]<< 32) | *lo*[*i*]

The following fast integer functions optimize the performance of kernels. In these functions, ***T*** is type int, int2, int3, int4, int8, int16, uint, uint2, uint4, uint8 or uint16.

T **mad24** (*T a*, *T b*, *T c*)	Multiply 24-bit int. values *a*, *b*, add 32-bit int. result to 32-bit int. *c*
T **mul24** (*T a*, *T b*)	Multiply 24-bit int. values *a* and *b*

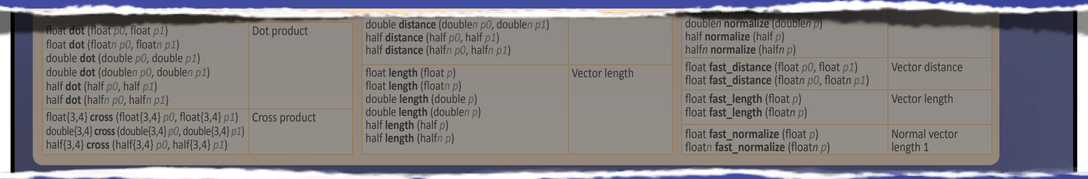
OpenCL API 1.1 Quick Reference Card - Page 3

Math Built-in Functions [6.11.2]

T is type float or float*n* (or optionally double, double*n*, or half*n*). int*n*, uint*n*, and ulong*n* must be scalar when *T* is scalar. ***Q*** is qualifier __global, __local, or __private. **HN** indicates that Half and Native variants are available by prepending "half_" or "native_" to function name. Prototypes shown in purple are half_ and native_ only. **Optional extensions enable double, double*n*, half, and half*n* types.**

T **acos** (*T*)	Arc cosine
T **acosh** (*T*)	Inverse hyperbolic cosine
T **acospi** (*T x*)	acos (*x*) / π
T **asin** (*T*)	Arc sine
T **asinh** (*T*)	Inverse hyperbolic sine
T **asinpi** (*T x*)	asin (*x*) / π
T **atan** (*T y_over_x*)	Arc tangent
T **atan2** (*T y*, *T x*)	Arc tangent of *y* / *x*
T **atanh** (*T*)	Hyperbolic arc tangent
T **atanpi** (*T x*)	atan (*x*) / π
T **atan2pi** (*T x*, *T y*)	atan2 (*x*, *y*) / π
T **cbrt** (*T*)	Cube root
T **ceil** (*T*)	Round to integer toward + infinity
T **copysign** (*T x*, *T y*)	*x* with sign changed to sign of *y*
T **cos** (*T*) **HN**	Cosine
T **cosh** (*T*)	Hyperbolic consine
T **cospi** (*T x*)	cos (π *x*)
T **half_divide** (*T x*, *T y*) *T* **native_divide** (*T x*, *T y*)	*x* / *y* (*T* may be float or float*n*)
T **erfc** (*T*)	Complementary error function
T **erf** (*T*)	Error function
T **exp** (*T x*)	e^*x*
T **exp2** (*T*)	2^*x*
T **exp10** (*T*)	10^*x*
T **expm1** (*T x*)	e^*x* -1.0
T **fabs** (*T*)	Absolute value
T **fdim** (*T x*, *T y*)	"Positive difference" between *x* and *y*
T **floor** (*T*)	Round to integer toward - infinity
T **fma** (*T a*, *T b*, *T c*)	Multiply and add, then round
T **fmax** (*T x*, *T y*) half*n* **fmax** (half*n x*, half *y*) float*n* **fmax**(float*n x*, float *y*) double*n* **fmax**(double*n x*, double *y*)	Return *y* if *x* < *y*, otherwise it returns *x*
T **fmin** (*T x*, *T y*) half*n* **fmin** (half*n x*, half *y*) float*n* **fmin**(float*n x*, float *y*) double*n* **fmin**(double*n x*, double *y*)	Return *y* if *y* < *x*, otherwise it returns *x*
T **fmod** (*T x*, *T y*)	Modulus. Returns *x* – *y* * trunc (*x*/*y*)
T **fract** (*T x*, *Q T* **iptr*)	Fractional value in *x*
T **frexp** (*T x*, *Q* int*n* **exp*)	Extract mantissa and exponent
T **hypot** (*T x*, *T y*)	Square root of *x*^2+ *y*^2
int*n* **ilogb** (*T x*)	Return exponent as an integer value
T **ldexp** (*T x*, int*n n*) *T* **ldexp** (*T x*, int *n*)	*x* * 2^*n*
T **lgamma** (*T x*) *T* **lgamma_r** (*T x*, *Q* int*n* **signp*)	Log gamma function
T **log** (*T*) **HN**	Natural logarithm
T **log2** (*T*) **HN**	Base 2 logarithm
T **log10** (*T*) **HN**	Base 10 logarithm
T **log1p** (*T x*)	ln (1.0 + *x*)
T **logb** (*T x*)	Exponent of *x*
T **mad** (*T a*, *T b*, *T c*) **HN**	Approximates *a* * *b* + *c*
T **maxmag** (*T x*, *T y*)	Maximum magnitude of *x*, *y*
T **minmag** (*T x*, *T y*)	Minimum magnitude of *x*, *y*
T **modf** (*T x*, *Q T* **ip*)	Decompose floating-point
float **nan** (uint*n nan*) float*n* **nan** (uint*n na* half*n* **nan** (ushort*n n* double*n* **nan** (ulong*n n*	Return a quiet NaN
T **nextafter** (*T x*, *T y*)	Next floating-point number
T **pow** (*T x*, *T y*)	*x*^*y*
T **pown** (*T x*, int*n y*)	*x*^*y*
T **powr** (*T x*, *T y*)	*x*^*y*
T **half_recip** (*T x*) *T* **native_recip** (*T x*)	1 / *x*
T **remainder** (*T x*, *T*	
T **remquo** (*T x*, *T y*, *Q* int*n* **quo*)	
T **rint** (*T*)	
T **rootn** (*T x*, int*n y*)	
T **round** (*T x*)	
T **rsqrt** (*T*)	
T **sin** (*T*)	
T **sincos** (*T x*, *Q T* *c	
T **sinh** (*T*)	
T **sinpi** (*T x*)	
T **sqrt** (*T*)	

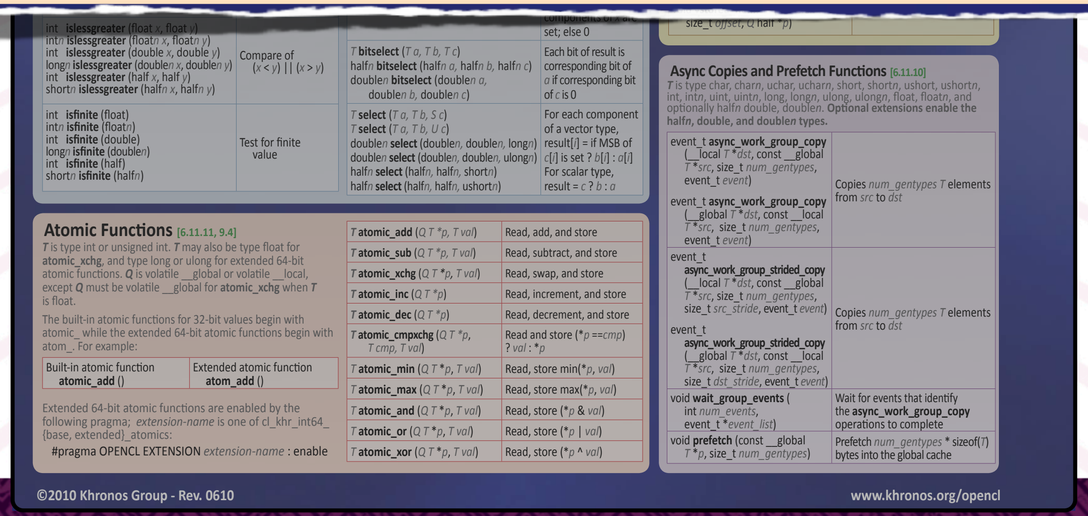
OpenCL API 1.1 Quick Reference Card - Page 4



Geometric Built-in Functions [6.11.5]

Vector types may have 2, 3, or 4 components. **Optional extensions enable double, double*n*, and half*n* types.**

float **dot** (float *p0*, float *p1*) float **dot** (float*n* *p0*, float*n* *p1*) double **dot** (double *p0*, double *p1*) double **dot** (double*n* *p0*, double*n* *p1*) half **dot** (half *p0*, half *p1*) half **dot** (half*n* *p0*, half*n* *p1*)	Dot product
float{3,4} **cross** (float{3,4} *p0*, float{3,4} *p1*) double{3,4} **cross** (double{3,4} *p0*, double{3,4} *p1*) half{3,4} **cross** (half{3,4} *p0*, half{3,4} *p1*)	Cross product
float **distance** (float *p0*, float *p1*) float **distance** (float*n* *p0*, float*n* *p1*) double **distance** (double *p0*, double *p1*) double **distance** (double*n* *p0*, double*n* *p1*) half **distance** (half *p0*, half *p1*) half **distance** (half*n* *p0*, half*n* *p1*)	Vector distance
float **length** (float *p*) float **length** (float*n* *p*) double **length** (double *p*) double **length** (double*n* *p*) half **length** (half *p*) half **length** (half*n* *p*)	Vector length
float **normalize** (float *p*) float*n* **normalize** (float*n* *p*) double **normalize** (double *p*) double*n* **normalize** (double*n* *p*) half **normalize** (half *p*) half*n* **normalize** (half*n* *p*)	
float **fast_distance** (float *p0*, float *p1*) float **fast_distance** (float*n* *p0*, float*n* *p1*)	
float **fast_length** (float *p*) float **fast_length** (float*n* *p*)	
float **fast_normalize** (float *p*) float*n* **fast_normalize** (float*n* *p*)	



Relational Built-in Functions [6.11.6]

T is type float, float*n*, char, char*n*, uchar, uchar*n*, short, short*n*, ushort, ushort*n*, int, int*n*, uint, uint*n*, long, long*n*, ulong, or ulong*n* (and optionally double, double*n*). ***S*** is type char, char*n*, short, short*n*, int, int*n*, long, or long*n*. ***U*** is type uchar, uchar*n*, ushort, ushort*n*, uint, uint*n*, ulong, or ulong*n*. **Optional extensions enable double, double*n*, and half*n* types.**

int **isequal** (float *x*, float *y*) int*n* **isequal** (float*n* *x*, float*n* *y*) int **isequal** (double *x*, double *y*) long*n* **isequal** (double*n* *x*, double*n* *y*) int **isequal** (half *x*, half *y*) short*n* **isequal** (half*n* *x*, half*n* *y*)	Compare of *x* == *y*
int **isnotequal** (float *x*, float *y*) int*n* **isnotequal** (float*n* *x*, float*n* *y*) int **isnotequal** (double *x*, double *y*) long*n* **isnotequal** (double*n* *x*, double*n* *y*) int **isnotequal** (half *x*, half *y*) short*n* **isnotequal** (half*n* *x*, half*n* *y*)	Compare of *x* != *y*
int **isgreater** (float *x*, float *y*) int*n* **isgreater** (float*n* *x*, float*n* *y*) int **isgreater** (double *x*, double *y*) long*n* **isgreater** (double*n* *x*, double*n* *y*) int **isgreater** (half *x*, half *y*) short*n* **isgreater** (half*n* *x*, half*n* *y*)	Compare of *x* > *y*
int **isgreaterequal** (float *x*, float *y*) int*n* **isgreaterequal** (float*n* *x*, float*n* *y*) int **isgreaterequal** (double *x*, double *y*) long*n* **isgreaterequal** (double*n* *x*, double*n* *y*) int **isgreaterequal** (half *x*, half *y*) short*n* **isgreaterequal** (half*n* *x*, half*n* *y*)	Compare of *x* >= *y*
int **isless** (float *x*, float *y*) int*n* **isless** (float*n* *x*, float*n* *y*) int **isless** (double *x*, double *y*) long*n* **isless** (double*n* *x*, double*n* *y*) int **isless** (half *x*, half *y*) short*n* **isless** (half*n* *x*, half*n* *y*)	Compare of *x* < *y*
int **islessequal** (float *x*, float *y*) int*n* **islessequal** (float*n* *x*, float*n* *y*) int **islessequal** (double *x*, double *y*) long*n* **islessequal** (double*n* *x*, double*n* *y*) int **islessequal** (half *x*, half *y*) short*n* **islessequal** (half*n* *x*, half*n* *y*)	Compare of *x* <= *y*
int **islessgreater** (float *x*, float *y*) int*n* **islessgreater** (float*n* *x*, float*n* *y*) int **islessgreater** (double *x*, double *y*) long*n* **islessgreater** (double*n* *x*, double*n* *y*) int **islessgreater** (half *x*, half *y*) short*n* **islessgreater** (half*n* *x*, half*n* *y*)	Compare of (*x* < *y*) || (*x* > *y*)
int **isfinite** (float) int*n* **isfinite** (float*n*) int **isfinite** (double) long*n* **isfinite** (double*n*) int **isfinite** (half) short*n* **isfinite** (half*n*)	Test for finite value
int **isinf** (float) int*n* **isinf** (float*n*) int **isinf** (double) long*n* **isinf** (double*n*) int **isinf** (half) short*n* **isinf** (half*n*)	
int **isnan** (float) int*n* **isnan** (float*n*) int **isnan** (double) long*n* **isnan** (double*n*) int **isnan** (half) short*n* **isnan** (half*n*)	
int **isnormal** (float) int*n* **isnormal** (float*n*) int **isnormal** (double) long*n* **isnormal** (double*n*) int **isnormal** (half) short*n* **isnormal** (half*n*)	
int **isordered** (float *x*, float *y*) int*n* **isordered** (float*n* *x*, float*n* *y*) int **isordered** (double *x*, double *y*)	

OpenCL API 1.1 Quick Reference Card - Page 5

Image Read and Write Built-in Functions [6.11.13, 9.5, 9.6.8]

The built-in functions defined in this section can only be used with image memory objects created with **clCreateImage2D** or **clCreateImage3D**. *sampler* specifies the addressing and filtering mode to use. **H** = To enable **read_imageh** and **write_imageh**, enable extension cl_khr_fp16.
3D = To enable type image3d_t in **write_image{f, i, ui}**, enable extension cl_khr_3d_image_writes.

float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) float4 **read_imagef** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) int4 **read_imagei** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) uint4 **read_imageui** (image2d_t *image*, sampler_t *sampler*, float2 *coord*)	Read an element from a 2D image
half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, int2 *coord*) **H** half4 **read_imageh** (image2d_t *image*, sampler_t *sampler*, float2 *coord*) **H**	
void **write_imagef** (image2d_t *image*, int2 *coord*, float4 *color*) void **write_imagei** (image2d_t *image*, int2 *coord*, int4 *color*) void **write_imageui** (image2d_t *image*, int2 *coord*, uint4 *color*)	Write *color* value to (*x*, *y*) location specified by *coord* in the 2D image
void **write_imageh** (image2d_t *image*, int2 *coord*, half4 *color*) **H**	
float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) float4 **read_imagef** (image3d_t *image*, sampler_t *sampler*, float4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) int4 **read_imagei** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	Read an element from a 3D image
uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, int4 *coord*) uint4 **read_imageui** (image3d_t *image*, sampler_t *sampler*, float4 *coord*)	
int **get_image_width** (image2d_t *image*) int **get_image_width** (image3d_t *image*)	
int **get_image_height** (image2d_t *image*) int **get_image_height** (image3d_t *image*)	
int **get_image_depth** (image3d_t *image*)	
int **get_image_channel_data_type** (image2d_t *image*) int **get_image_channel_data_type** (image3d_t *image*)	
int **get_image_channel_order** (image2d_t *image*) int **get_image_channel_order** (image3d_t *image*)	
int2 **get_image_dim** (image2d_t *image*)	
int4 **get_image_dim** (image3d_t *image*)	
Use this pragma to enable type image3d_t in **write_image{f, i, ui}**: #pragma OPENCL EXTENSION cl_khr_3d_image_writes : enable	
void **write_imagef** (image3d_t *image*, int4 *coord*, float4 *color*) **3D**	
void **write_imagei** (image3d_t *image*, int4 *coord*, int4 *color*) **3D**	
void **write_imageui** (image3d_t *image*, int4 *coord*, uint4 *color*) **3D**	

Map and Unmap Image Objects [5.3.5]

void * **clEnqueueMapImage** (
cl_command_queue *command_queue*, cl_mem *image*,
cl_bool *blocking_map*, cl_map_flags *map_flags*,
const size_t *origin*[3], const size_t *region*[3],
size_t *image_row_pitch*, size_t *image_slice_pitch*,
cl_uint *num_events_in_wait_list*, const cl_event *event_wait_list,
cl_event *event, cl_int *errcode_ret)

Access Qualifiers [6.6]

Apply to image image2d_t and image3d_t types to declare if the image memory object is being read or written by a kernel. The default qualifier is __read_only.

__read_only, read_only
__write_only, write_only

cl_int **clGetSamplerInfo** (cl_sampler *sampler*,
cl_sampler_info *param_name*,
size_t *param_value_size*, void *param_value,
size_t *param_value_size_ret)

param_name: CL_SAMPLER_REFERENCE_COUNT,
CL_SAMPLER_{CONTEXT, FILTER_MODE},
CL_SAMPLER_ADDRESSING_MODE,
CL_SAMPLER_NORMALIZED_COORDS

OpenCL Course:

14:15-18:00 Room 308B-C

- Development Tips
- Graphics Interop
- Physical Simulations
- Visualisation Applications

Questions?

Derek Gerstmann

University of Western Australia

http://local.wasp.uwa.edu.au/~derek