## Vector Programming Overview

- Execution Models for GPU Architecture: MIMD, SIMD, SIMT


Figure 12: Fundamental Design Philosophy of CPU vs GPU

- **Control Logic**: Allow Parallel and/or Out-of-Order execution of threads. (Centralized on CPU, Decentralized on GPU)
- **ALU**: Perform arithmetic and bitwise operations (One ALU for each core on CPU, One ALU for each core on GPU, or One Arithmetic Unit (FPU) for each core)
- **Cache**: On-chip Memory to reduce instruction and data access latencies (Very small capacity on GPU)

## Vector Programming Overview (cont.)

- **DRAM CPU**: Off-chip Memory to store different processes

**Why are people switching to GPU's?**

- Performance Reasons (Offload numerically intensive parts to GPU)
- Processor availability in Market (Program for the dominant processor. 10 years ago, parallel parallel computing limited to governments and large corporations/universities. This has all changed now with GPUs, thanks to video games.)
- Massive scalability in limited space (Embedded applications requiring parallelism could not include large cluster-based machines. With GPUs, they can)
- IEEE Floating Point Compliancy (Early GPU's were not entirely IEEE compliant. Hence programmers refrained to use them. This is now almost history, unless you buy an old GPU)
- Graphics Programming no longer required to operate on Graphics Cards. We have **GPGPU** compliant API's.

## Example: NVIDIA Kepler K40

- GP-GPU, Scientific Computing
- Slave Processors
- GPU Giants (NVIDIA + AMD)
- CUDA: NVIDIA based GPU's
- OpenCL: Open coding standard for cross-device execution (Mobile Phones, GPU, CPU, Altera FPGA's), established by Khronos Group (2008)

| - | Kepler K40 | Intel i7-4900MQ |
|---|---|---|
| Cores | 2,880 | 4 (8 Threads) |
| RAM | 12 GB | - |
| Cache | 48 Kb | 8 Mb |
| Clock | 876 MHz | 2.8 GHz |
| Bwidth | 288 GB/s | 25.6 GB/s |
| FLOP (d) | 1.40 TFLOP | 13.97 GFLOP |
| FLOP (f) | 4.29 TFLOP | 25.76 GFLOP |

NVIDIA k40, 2880 cores, 12 GB RAM

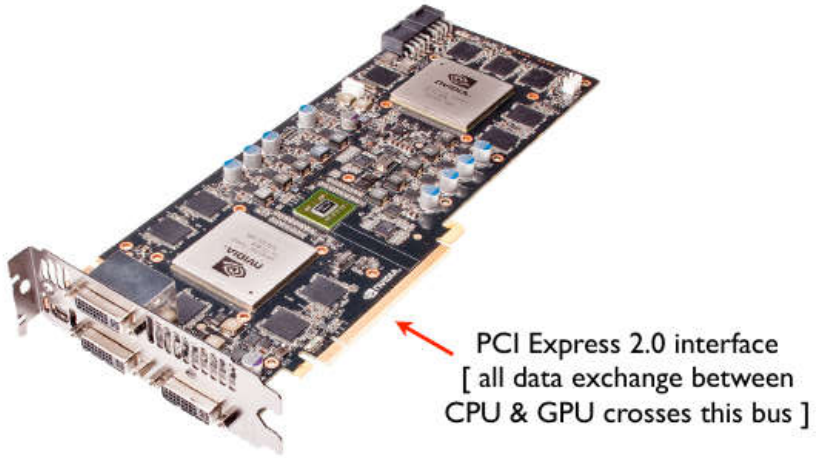
# Example: NVIDIA GTX 590


GTX 590 GPU

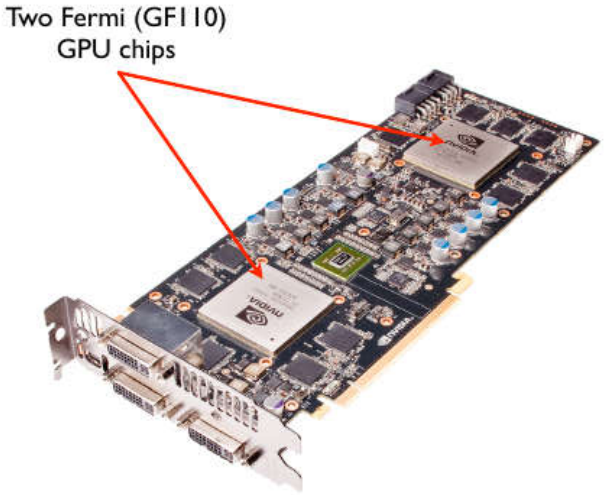

GTX 590 GPU

- Cores: 1024, Processor Clock: 1215 MHz, Memory: 3 GB

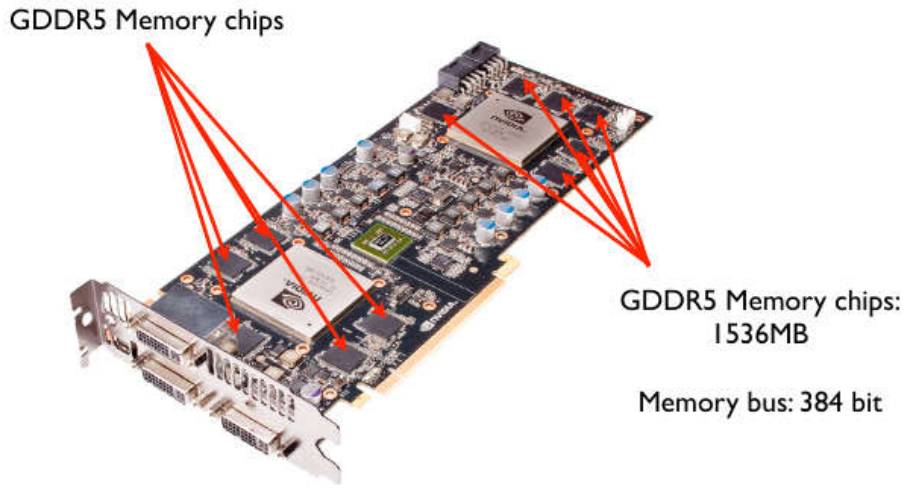
# Example: NVIDIA GTX 590 (cont.)



# Example: NVIDIA GTX 590 (cont.)



## Example: NVIDIA GTX 590 (cont.)



## Example: NVIDIA GTX 590 (cont.)

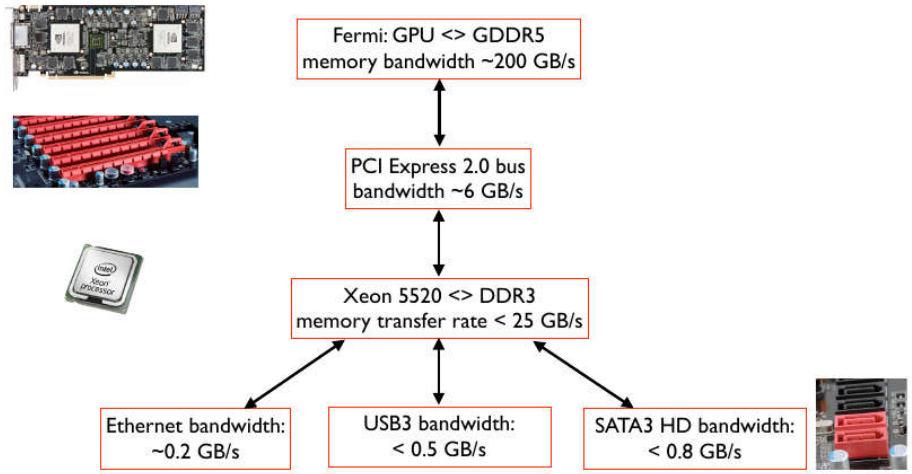



Figure 13: The Bottleneck

# Trying it Out

## Access Details

`ssh 121.52.146.108 -l cdc-username -XY`

where, **username** can be provided on request
and **password** is same as username (one time only)

Example session with password = p116003 would be:

`ssh 121.52.146.108 -l cdc-p116003 -XY`

## Introduction to GPGPU's

### Open Computing Language (**OpenCL**)

- An **open** standard from **Khronos**; the makers of OpenGL (v1.0 Release December 2008)
- Cross Platform/Vendor/Architecture (CPU, GPU, DSP, FPGA, ...)
- GPU Giants (NVIDIA + AMD)
- Other Major Players (Apple, Intel, Qualcomm, Samsung, Xilinx, Altera)
- OpenCL is a {Standard, Language (based on C99), API/Library, Runtime SIMT based Compilation and Execution Environment}
- Two-way inter-operatable with OpenGL

### Compute Unified Device Architecture (**CUDA**)

- **Proprietary** platform/API, released by **NVIDIA** (v1.0 released in January 2007)
- Handles only one platform/vendor, i.e., NVIDIA manufactured Graphic Cards
- CUDA is a {Language, API/Library, Non-runtime compilation environment, and Runtime execution environment}
- Inter-operatability with OpenGL is one-way (OpenGL can view CUDA buffers, but CUDA cannot view OpenGL buffers)

## Introduction to GPGPU's (cont.)

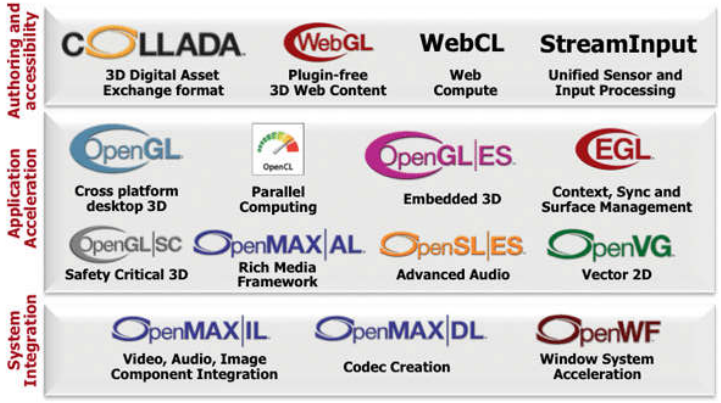
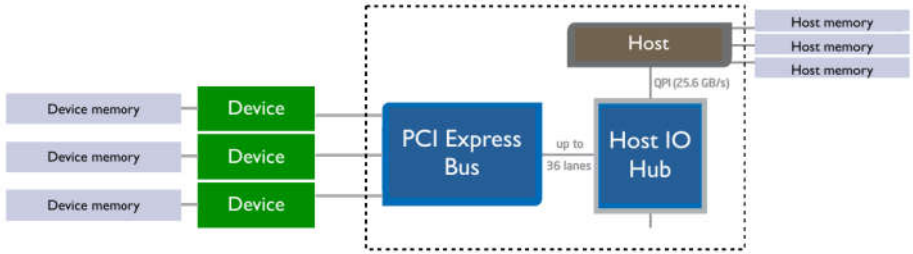



Figure 14: Khronos Group Open Specifications

[Img] http://www.khronos.org/about

# Data Migration



# Data Migration (cont.)

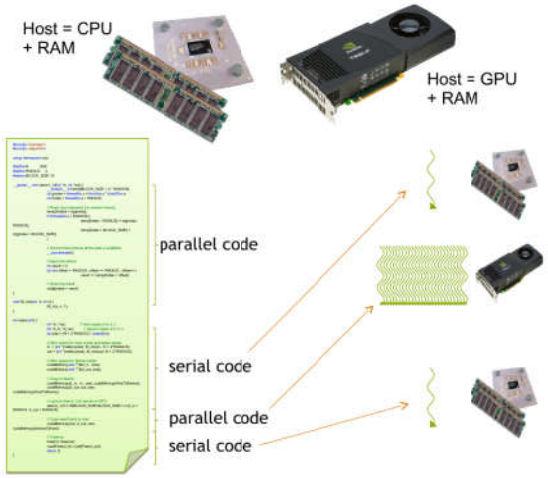
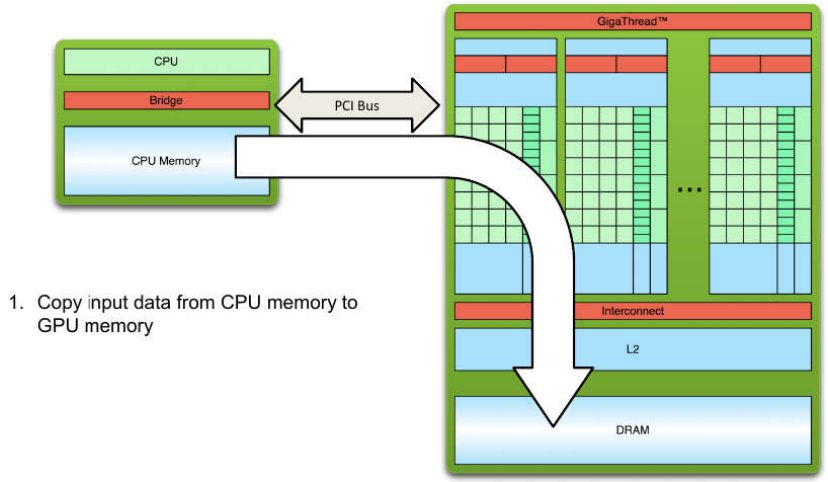


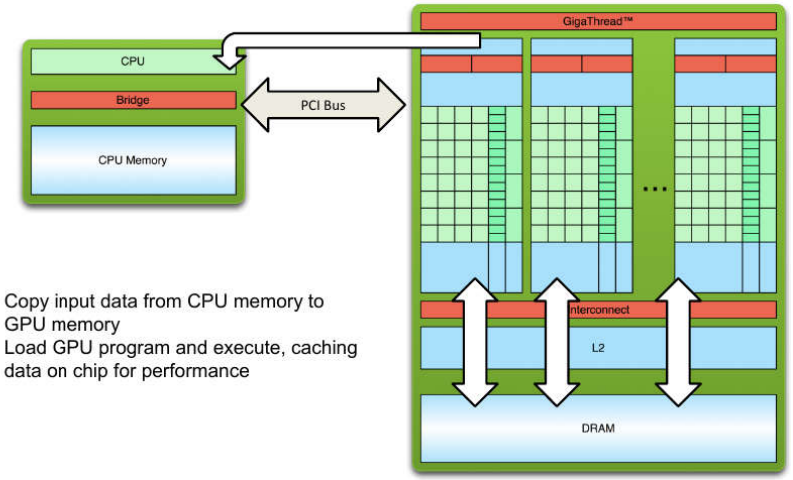

Figure 15: Porting portions of your code to GPU

# Data Migration (cont.)



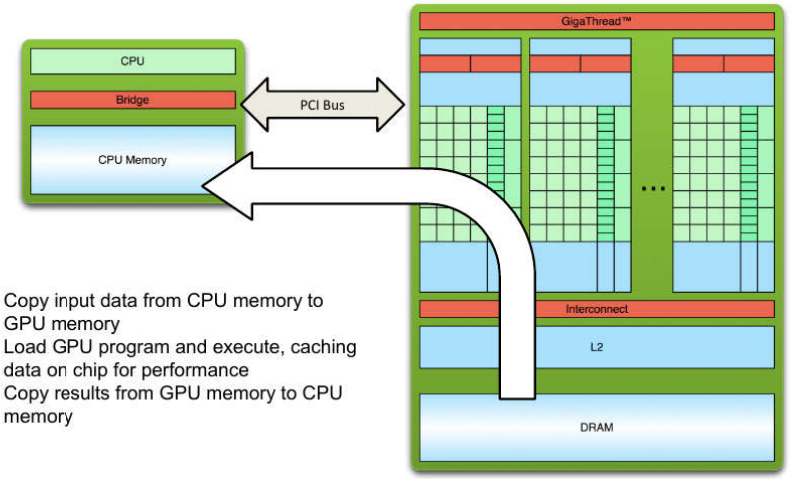

1. Copy input data from CPU memory to GPU memory

# Data Migration (cont.)



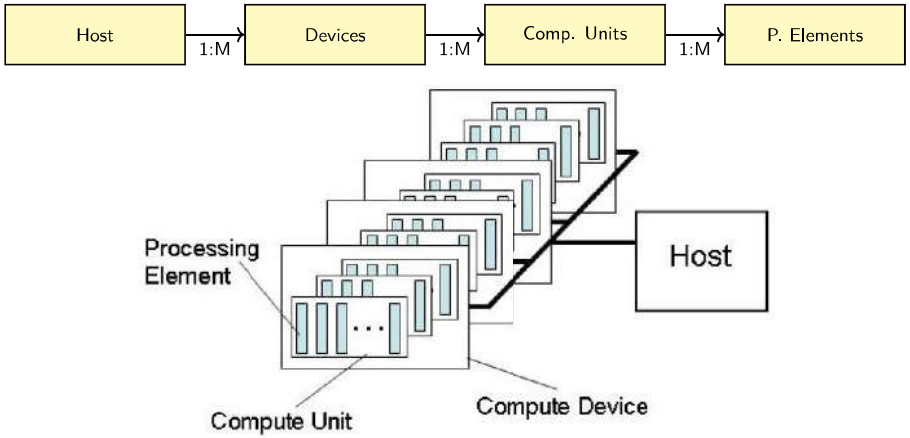

1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

# Data Migration (cont.)




1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

# OpenCL Specification: Platform Model



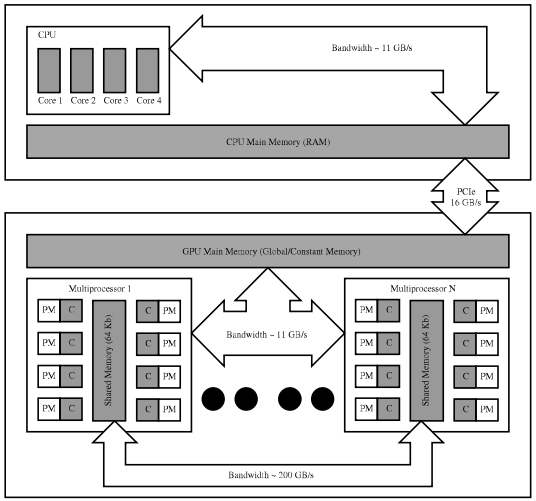

- NVIDIA-SDK: oclDeviceQuery

## OpenCL Specification: Platform Model (cont.)

| NVIDIA/CUDA | AMD/OpenCL |
|---|---|
| CUDA Core | Processing Element |
| Streaming Multiprocessor | Compute Unit |
| GPU Device | Compute Device |

Table 4: Terminology Difference between CUDA and OpenCL

# Memory Model




## Global Memory

- Permits R/W access to all work-items in all work-groups
- May be cached (hardware dependent)

## Constant Memory

- Belongs to same address space as global memory
- Host initializes and places contents into it
- Contents remain constant (R/O) to work-items

# Memory Model (cont.)

## Local (Shared) Memory

- Can be used to allocate variables that are shared by all work-items in a particular work-group.
- Have much faster access speed than global memory (./oclBandwidthTest)

## Private Memory (Registers)

- Can be used to allocate variables for a given work-item
- Visibility scope belongs to work-item only

## Memory Address Space Specifiers

| Declaration (OpenCL) | Declaration (CUDA) | Memory | Scope | Lifetime |
|---|---|---|---|---|
| __private__ | none | register | thread | kernel |
| __local__ | __shared__ | shared | block | kernel |
| __global__ | __device__ | global | grid | application |
| __constant__ | __constant__ | constant | grid | application |

# Memory Model (cont.)

```
CL_DEVICE_MAX_MEM_ALLOC_SIZE:       2879 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:          11519 MByte
CL_DEVICE_LOCAL_MEM_SIZE:           48 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE: 64 KByte
CL_DEVICE_REGISTERS_PER_BLOCK_NV:   65536
```

## OpenCL Specification: Execution Model

**Host** Cuda/OpenCL applications have a single host program (that runs on the CPU. It is responsible for preparing the execution environment for the GPU.

**Kernel** A host program launches a number of instances of a kernel.

**Command Queue** All kernel management related tasks (performed by the CPU) such as memory transfer from device RAM to CPU RAM, from CPU RAM to device RAM, launching kernels, etc. are placed as commands on the command queue.

**Work Item** Each instance of a kernel is known as a work-item.

**Work Group** Arrangements of work-items into logical groups by developer.

**Warp** Arrangements of work-items into physical groups by thread scheduler.

**Index Space** An N-Dimensional range of values (N can only be 1, 2, or 3) that is used for identifying a thread uniquely. Range of values represented by integer arrays of length N.

## OpenCL Specification: Execution Model (cont.)

| NVIDIA/CUDA | AMD/OpenCL |
|---|---|
| Kernel | Kernel |
| Stream | Command Queue |
| Thread | Work Item |
| Block | Work Group |
| Warp | Warp |
| Grid | NDrange |

Table 5: Execution Model Terminology Differences between CUDA and OpenCL

# OpenCL Specification: Execution Model (cont.)

## Index Space

**Global Size** Size of NDRange integer array along each dimension $(G_x, G_y, G_z)$

**Global ID** Identify of a thread in global context along each dimension $(g_x, g_y, g_z)$. Thus, total range of threads would be $[0, 1, \ldots, G_x - 1], [0, 1, \ldots, G_y - 1], [0, 1, \ldots, G_z - 1]$

**Group Size** Total number of work-groups along each dimension $(W_x, W_y, W_z)$

**Group ID** Identify of each work-group along each dimension $(w_x, w_y, w_z)$

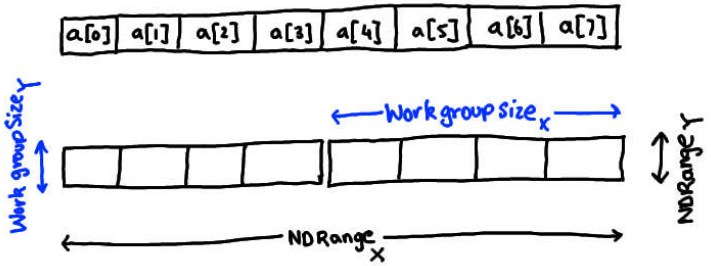
**Local Size** Size of work-group along each dimension $(L_x, L_y, L_z)$. For GPU programming, the Local Size must **evenly** divide the Global Size.

```
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:      3
CL_DEVICE_MAX_WORK_ITEM_SIZES:      1024 / 1024 / 64 (512/512/32)
CL_DEVICE_MAX_WORK_GROUP_SIZE:      1024 (512)
```

**Local ID** Identify of a thread in local context (of work-group) along each dimension. Thus, total range of threads would be $[0, 1, \ldots, L_x - 1], [0, 1, \ldots, L_y - 1], [0, 1, \ldots, L_z - 1]$ for each work-group.

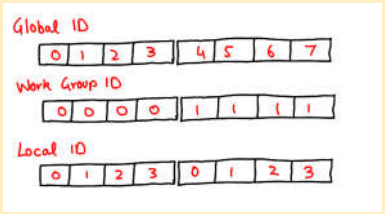
# OpenCL Specification: Execution Model (cont.)

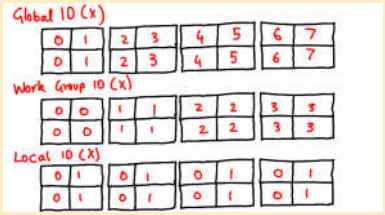
```
int get_work_dim();      // Return number of dimensions
int get_num_groups(int); // Return number of work-groups W_{x,y,z}
int get_global_size(int); // Return NDRange Size G_{x,y,z}
int get_local_size(int);  // L_{x,y,z} = G_{x,y,z}/W_{x,y,z}
int get_global_id(int);   // g_{x,y,z} = w_{x,y,z} × L_{x,y,z} + l_{x,y,z}
                          // g_{x,y,z} = w_{x,y,z} × L_{x,y,z} + l_{x,y,z} + o_{x,y,z}
int get_group_id(int);    // w_{x,y,z} = g_{x,y,z}/L_{x,y,z}
int get_local_id(int);    // l_{x,y,z} = g_{x,y,z}%L_{x,y,z}
```

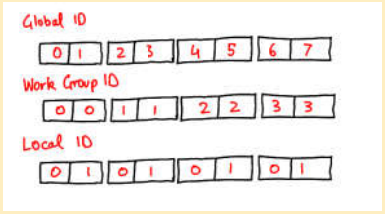


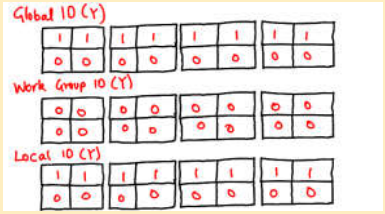
# OpenCL Specification: Execution Model (cont.)

## Examples of index Space



# OpenCL Specification: Execution Model (cont.)

## The OpenCL Context

The execution environment created by host for executing the kernels. Comprises of:

**Devices** Collection of OpenCL devices to be used by the host

**Kernels** OpenCL functions that would run on OpenCL devices

**Program Objects** The kernel source code and/or executables. These will be built at runtime **Within the Host Program.** Source code can be specified through three means:

- Regular string definition within code
- Read line-by-line from file
- **generated** dynamically inside the host program through string manipulation

**Memory Objects** Sets of objects in memory that are visible to OpenCL devices and contain values that can be be operated upon by instances of kernels.

# OpenCL Specification: Execution Model (cont.)

## Command Queue

- Interaction between host and OpenCL device occurs through commands posted on the command queue. Types of commands supported:
  - Kernel execution commands
  - Memory transfer commands (from device RAM to CPU RAM, and CPU RAM to device RAM)
  - Synchronization commands
- Multiple command queues can be created for doing things simultaneously.
- Typical command sequence (copy from CPU RAM to Device RAM, Execute Kernel, copy from Device RAM to CPU RAM)
- **In-Order Execution**: Commands launched in order in which they appear in command queue. New command only starts when old one is completed.
- **Out-of-Order Execution**: Commands placed in order, but new commands can start executing even if previous one is not yet completed.

# OpenCL Specification: Memory Model

## Memory Objects

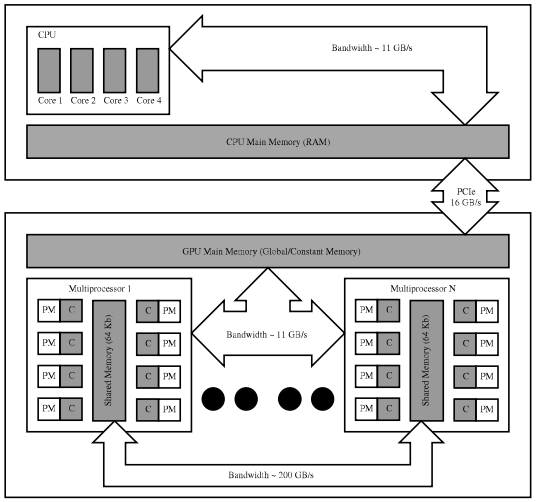
**Buffer Objects** Contiguous block of memory (basically an array).

- Can be present in either host RAM, or device RAM
- Can be accessed using pointers
- Can be read-only, or write-only, or read-write
- May be cached (hardware dependent)

**Image Objects** Used for storing images

- Can be present in GPU Texture Memory
- Can be accessed as values
- Can be read-only, or write-only
- Can be cached (irrespective of hardware)

# OpenCL Specification: Memory Model




## Global Memory

- Permits R/W access to all work-items in all work-groups
- May be cached (hardware dependent)

## Constant Memory

- Belongs to same address space as global memory
- Host initializes and places contents into it
- Contents remain constant (R/O) to work-items

# OpenCL Specification: Memory Model (cont.)

## Local (Shared) Memory

- Can be used to allocate variables that are shared by all work-items in a particular work-group.
- Have much faster access speed than global memory (./oclBandwidthTest)

## Private Memory

- Can be used to allocate variables for a given work-item
- Visibility scope belongs to work-item only

```
CL_DEVICE_MAX_MEM_ALLOC_SIZE:       2879 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:          11519 MByte
CL_DEVICE_LOCAL_MEM_SIZE:           48 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE: 64 KByte
CL_DEVICE_REGISTERS_PER_BLOCK_NV:   65536
```