Open CL

OpenCL is an open parallel programming standard that allows you to create applications for simultaneous execution on many cores of modern processors, different in architecture, in particular, graphic (GPU) or central (CPU).

In other words, OpenCL allows you to use all the cores of the central processor or all the computing power of the video card for computing one task, which ultimately reduces the program execution time. Therefore, the use of OpenCL is very useful for computationally intensive tasks, but it is important to note that the algorithms for solving these tasks must be divisible into parallel threads. These include, for example, training neural networks, Fourier transform, or solving systems of equations of large dimensions.

OpenCL (Open Computing Language) is framework that allows users to write programs to execute across CPU (Central Processing Unit), GPU (Graphics Processing Unit), or dedicated accelerator device with benefit that it can speed up heavy computation required as per problem domain. Especially using GPU with OpenCL as GPU is designed to handle large amounts of data in parallel with high memory bandwidth and dedicated instruction set which is optimized in mathematics calculations. It has lots of processing core called compute units each of which can independently carry on the computation. In contrast, CPU is designed to carry out more general task, not specialized in the same sense as GPU. It has less processing cores. In short, CPU is suitable for different task than GPU. GPU is much faster for heavy duty computation especially in regards to graphics domain.

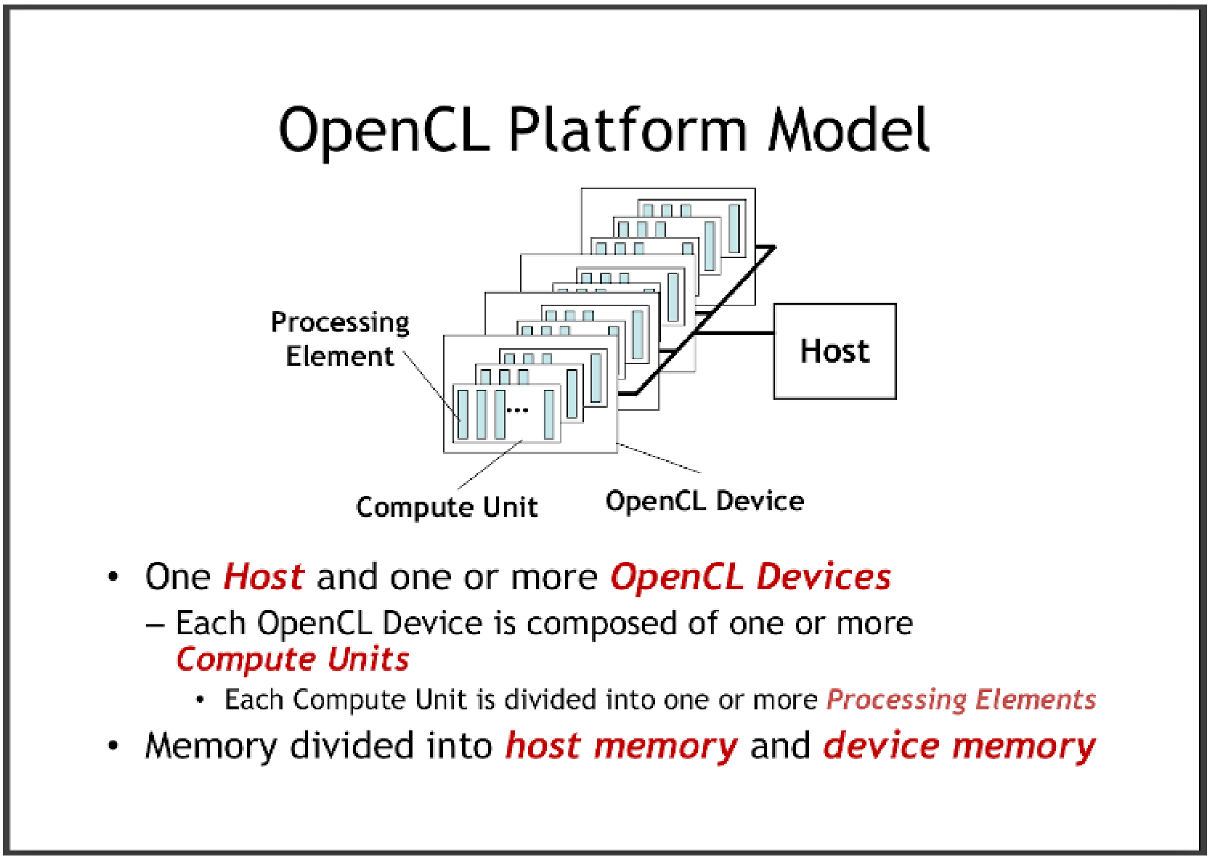
OpenCL device drivers automate the distribution of calculations across cores. For example, if you need to perform a million of calculations of the same type with different vectors, and there are only a thousand cores at your disposal, then the drivers will automatically start each next task as the previous ones are ready and the cores are released.

The OpenCL Platform Model

The platform model of OpenCL is similar to the one of the CUDA programming model. In short, according to the OpenCL Specification, "The model consists of a host (usually the CPU) connected to one or more OpenCL devices (e.g., GPUs, FPGAs). An OpenCL device is divided into one or more compute units (CUs) which are further divided into one or more processing elements (PEs). Computations on a device occur within the processing element"

An OpenCL program consists of two parts: host code and device code. As the name suggests, the host code is executed by the host and also "submits the kernel code as commands from the host to OpenCL devices".

Finally, such as in the CUDA programming model, the host communicates with the device(s) through the global memory of the device(s). As in the CUDA programming model, there is a memory hierarchy on the device. However, we have omitted these details for the sake of greater simplicity.



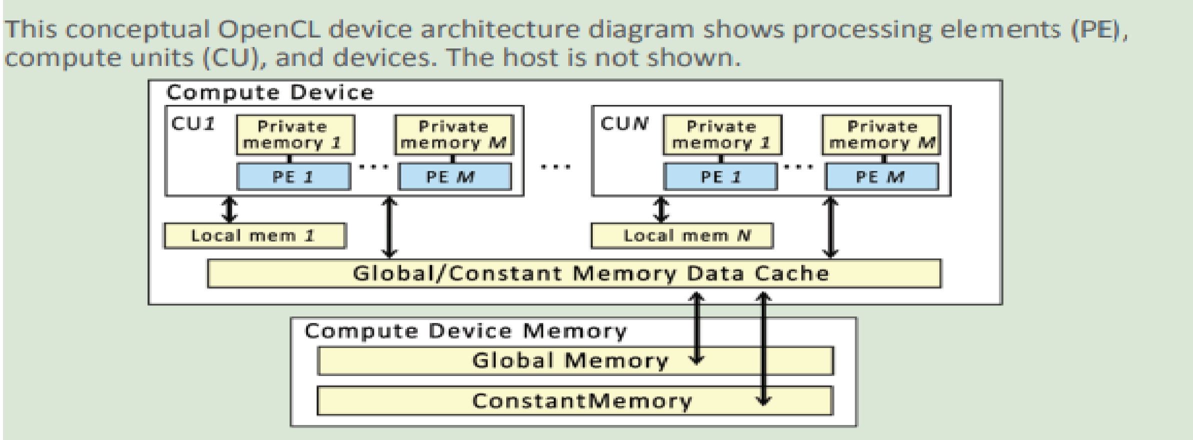
OpenCL Platform Model

- One *Host* and one or more *OpenCL Devices*
  - Each OpenCL Device is composed of one or more *Compute Units*
    - Each Compute Unit is divided into one or more *Processing Elements*
- Memory divided into *host memory* and *device memory*

OpenCL Platform Model - Courtesy of OpenCL: A Hands-on Introduction (Tim Mattson, Alice Koniges, and Simon McIntosh-Smith)

As we can see from the figure, OpenCL device (think GPU) consists of bunch of Compute unit. Each of it consists of dozens of Processing Element (PE). At the big picture, memory divided into host memory, and device memory.

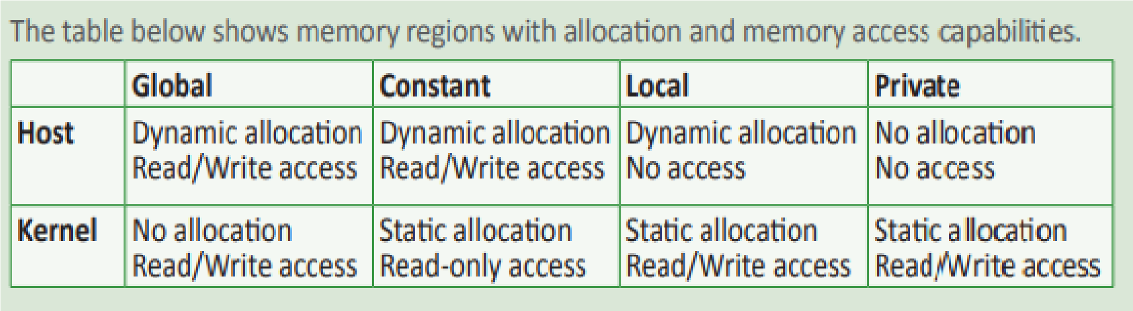
OpenCL Device Architecture Diagram



This conceptual OpenCL device architecture diagram shows processing elements (PE), compute units (CU), and devices. The host is not shown.

OpenCL Device Architecture Diagram - Courtesy of OpenCL API 1.2 Reference Guide by Khronos

Figure above zooms in further and gives us another perspective look on the same model we've seen previously but with additional of memory model layout along with their corresponding capabilities. Each PE has a private memory for fast access before requiring to communicate with local memory, and global/constant cache. Constant memory is fixed (read-only) memory type that is expected not to change during the course of kernel execution. Higher up from cache memory is the main memory of the device itself which requires more latency to get access there. Also as seen, there are similar global/constant memory type in the main memory of the device.

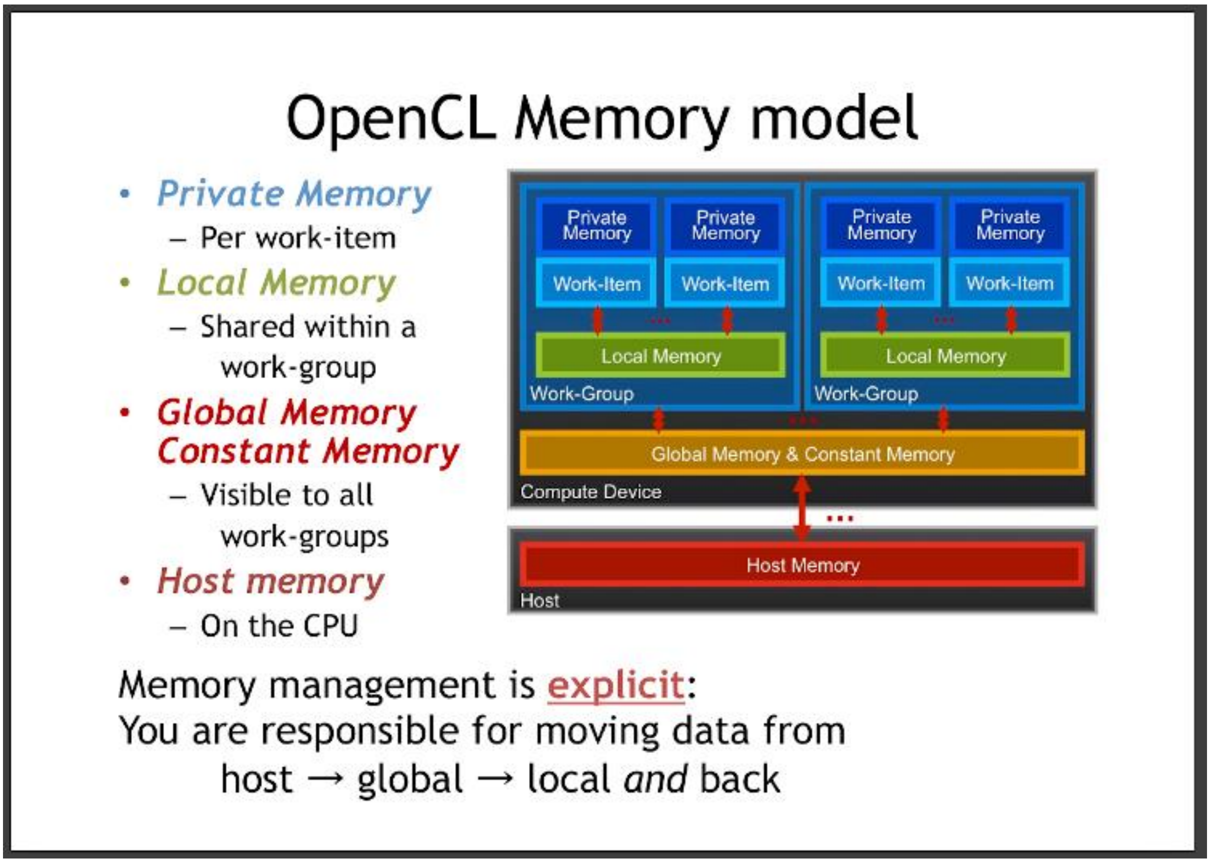


The table below shows memory regions with allocation and memory access capabilities.

	Global	Constant	Local	Private
Host	Dynamic allocation Read/Write access	Dynamic allocation Read/Write access	Dynamic allocation No access	No allocation No access
Kernel	No allocation Read/Write access	Static allocation Read-only access	Static allocation Read/Write access	Static allocation Read/Write access

OpenCL Device Architecture Diagram - Courtesy of OpenCL API 1.2 Reference Guide by Khronos

OpenCL Memory Model



OpenCL Memory model

- *Private Memory*
  - Per work-item
- *Local Memory*
  - Shared within a work-group
- *Global Memory*
  Constant Memory
  - Visible to all work-groups
- *Host memory*
  - On the CPU

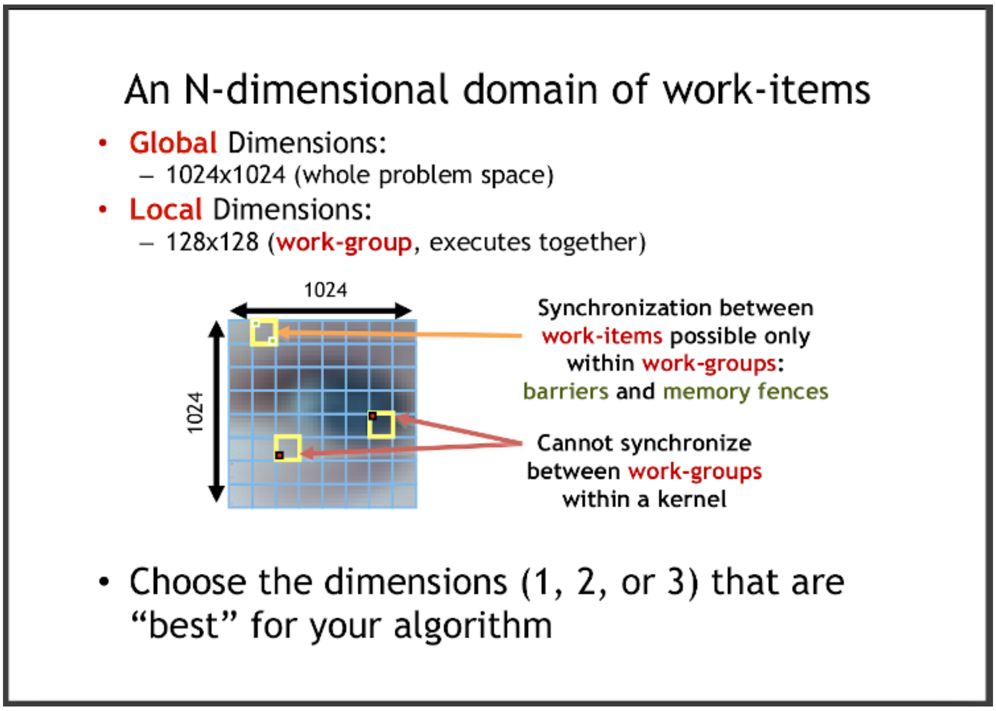
Memory management is explicit:
You are responsible for moving data from
host → global → local *and* back

OpenCL Memory Model - Courtesy of OpenCL: A Hands-on Introduction (Tim Mattson, Alice Koniges, and Simon McIntosh-Smith)

See the above figure for another clearer overview of memory model with some interchangeably terms. Similarly, but with notable key terms on work-item, and work-group. We can see PE as work-item. There are lots of work-items as per single work-group. Memory model as previously mentioned dispersed all across the whole architecture working from work-item to work-group, and interconnecting between device and host (think PC). Notable note as seen at the bottom of the figure is that we as a user of OpenCL would be responsible for moving data back and forth between host and device. We will see why this is the case when we get involved with the code. But in short, because data on both ends need to be synchronized for consistency in consuming result from computation or feeding data for kernel execution.

N-Dimensional domain of work-items

Let's see yet another layout of how work-item, and work-group would actually mean in context of the work to be carried out.



An N-dimensional domain of work-items

- Global Dimensions:
  - 1024x1024 (whole problem space)
- Local Dimensions:
  - 128x128 (work-group, executes together)

1024
1024
Synchronization between work-items possible only within work-groups: barriers and memory fences
Cannot synchronize between work-groups within a kernel

- Choose the dimensions (1, 2, or 3) that are "best" for your algorithm

Relationship between work-items and work-groups in context of kernel to execute - Courtesy of OpenCL: A Hands-on Introduction (Tim Mattson, Alice Koniges, and Simon McIntosh-Smith)

From figure above, that depicts imaginary work to be done in terms of problem space. Understand a problem space, will allow us to know global dimensions and potentially leads us to know local dimensions as well. Such dimension is one of important configurations to be efficient in working with OpenCL API. API itself limits to use at max 3 dimensions (3-dim), so if our problem space needs more than that, then we have to translate such problem into 3-dim problem. Image processing is the prime example here whose global dimensions can be set to 2-dim for WxH which expresses width and height of the image. We can deduce local dimensions to best fit the capability of device we will be using to carry out such task, but mostly value of each global dimension is divisible by corresponding value in local dimension. But how to know such value for local dimension? No worry, we can query device's capability with OpenCL API. We don't have to do a guess work. We can implement a generic solution that works equally the same adapting for all range of GPU device's capability.

As mentioned previously, a single work-group consists of several work-items. Each work-item will be performed by a single thread inside a device itself. A device especially GPU is able to spawn tons of threads to carry out the computation in parallel; imagine each work-group is computed in parallel to others but by how many work-group to be performed in parallel at the time is up to the GPU's capability.

OpenCL Terminology Summary

Key Term	Explanation
Host	A system that runs OpenCL application and communicates with OpenCL device. The host is responsible for preparation, and management work prior to execution of kernel. It can be PC, workstation, or cluster.
Device	A component in OpenCL that executes OpenCL kernel. It can be CPU, GPU, Accelerator, or custom one.
Kernel	A function written in OpenCL C language in which it will perform mathematical calculation for us on devices that OpenCL supports
Work-item	A unit of execution within a kernel. Collection of work-items are individual instances of the kernel that executed in parallel. Multiple work-items are called work-group.
Work-group	A group of work-items that executed in parallel. It is a logical group in which inside can share data and synchronize their execution
Private memory	A memory region that is specific to a work-item. Each work-item has its own private memory, and it is not shared between work-items. It is intended to store local variables, and temporary data which are only needed by a single work-item.
Local memory	A memory region that shared among work-items within a same work-group. Thus allow work-items to communicate. It is faster to access than global/constant memory.
Global memory/cache	A shared memory region that can be accessed by work-group (thus work-items). Cache would be located near the processing unit, but memory is far away (analogy to CPU cache, and RAM respectively).
Constant memory/cache	A shared memory region which is intended for read-only (data is not supposed to be changed during execution) that can be accessed by work-group (thus work-items). Cache would be located near the processing unit, but memory is far away (analogy to CPU's instruction cache, and ROM).
Global dimension	A configuration as value and number of dimensions e.g. 1024x1024 (2 dimensions with 1024 in each dimension) to describe a problem space. Maximum at 3 dimensions.
Local dimensions	A configuration as value and number of dimensions e.g. 128x128 (2 dimensions with 128 in each dimension) to describe number of work-items and work-groups to execute a kernel. For example, in case of global dimensions as of 1024x1024, and local dimensions of 128x128,

Key Term	**Explanation**
	number of work-items for a single work-group are 128*128=16,384, number of work-group needed is 1024/128 * 1024/128 = 8*8 = 64, total work-item is 16,384*64 or 1024*1024 =1,048,576.
Processing Element (PE)	A physical computing unit for support device e.g. CPU or GPU.
Compute Unit	A collection of PE within an OpenCL device that can execute multiple threads in parallel.