OpenCL (Open Computing Language)

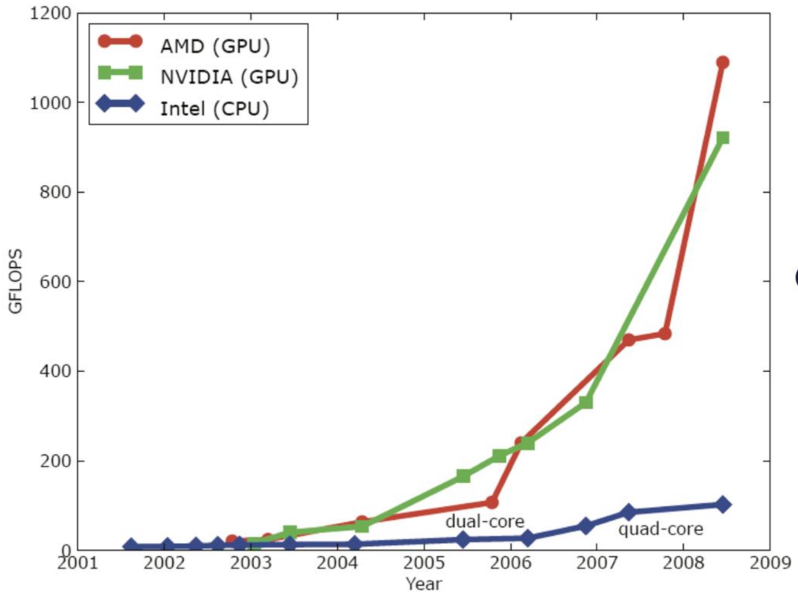
is a framework for writing programs that execute across heterogeneous platforms

Е.М. Нанивская
НУГ геоинформатики
http://geolab.gis.land/

History of OpenCL. Why GPU Computing

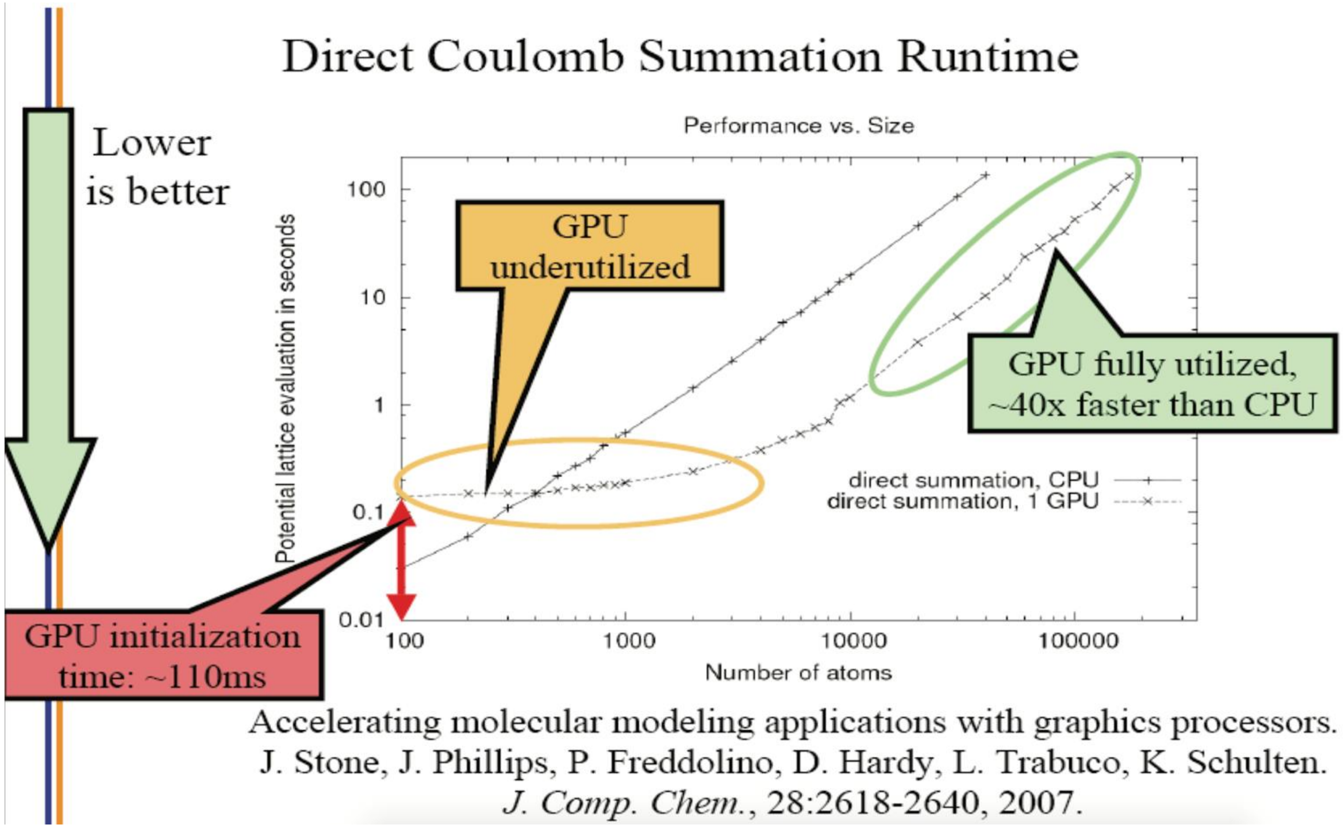
An enlarging peak performance advantage:

- Calculation: 1 TFLOPS vs. 100 GFLOPS
- Memory Bandwidth: 100-150 GB/s vs. 32-64 GB/s
- GPU in every PC and workstation – massive volume and potential impact



Courtesy: John Owens

History of OpenCL. Role of GPUs - large data sets



Accelerating molecular modeling applications with graphics processors. J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten. *J. Comp. Chem.*, 28:2618-2640, 2007.

History of OpenCL. Future Apps Reflect a Concurrent World

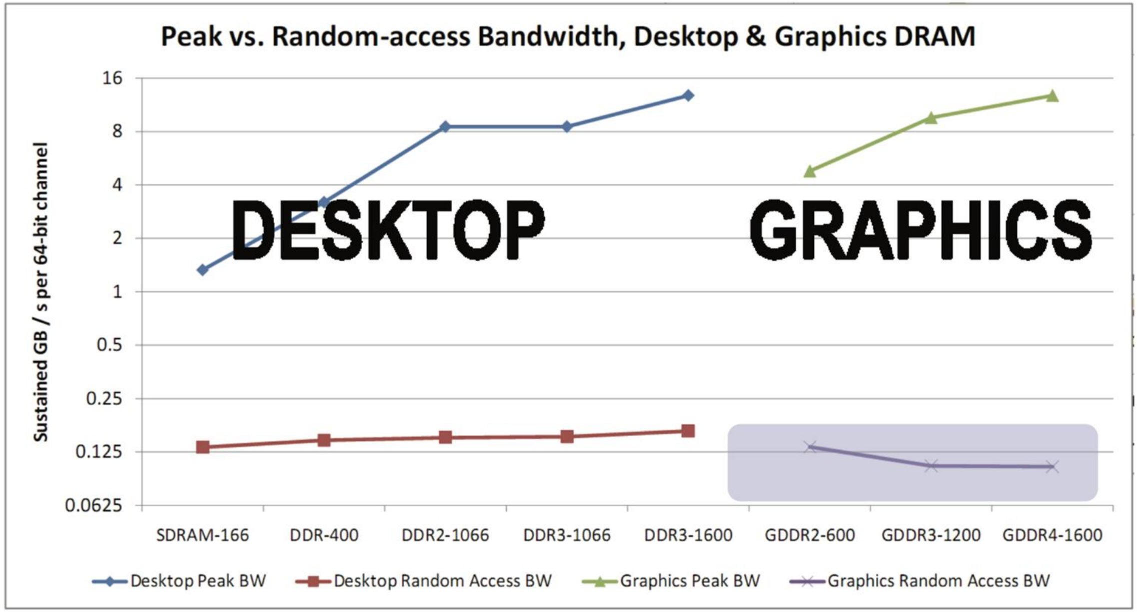
- Excitingapplicationsinfuturecomputinghavebeen traditionally considered "supercomputing applications"
  - Video and audio synthesis/analysis, 3D imaging and visualization, consumer game physics, virtual reality products, computational financing, molecular dynamics simulation, computational fluid dynamics
  - These "Super-apps" represent and model the physical, concurrent world

- Various granularities of parallelism exist, but...
  - programming model must not hinder scalable implementation
  - data delivery needs careful management

DRAM Bandwidth Trends Sets Programming Agenda

- Random access BW 1.2% of peak for DDR3-1600, 0.8% for GDDR4-1600 (and falling)

- 3D stacking and optical interconnects will unlikely help.



History of OpenCL. UIUC/NCSA AC Cluster

- 32 nodes UIUC/NCSA AC Cluster
  - 4-GPU (GTX280, Tesla), 1-FPGA, quad-core Opteron node at NCSA
  - GPUs donated by NVIDIA
  - FPGA donated by Xilinx
  - 128 TFLOPS single precision, 10 TFLOPS double precision
- Coulomb Summation:
  - 1.78 TFLOPS/node
  - 271x speedup vs. Intel QX6700 CPU core w/ SSE

UIUC/NCSA AC Cluster

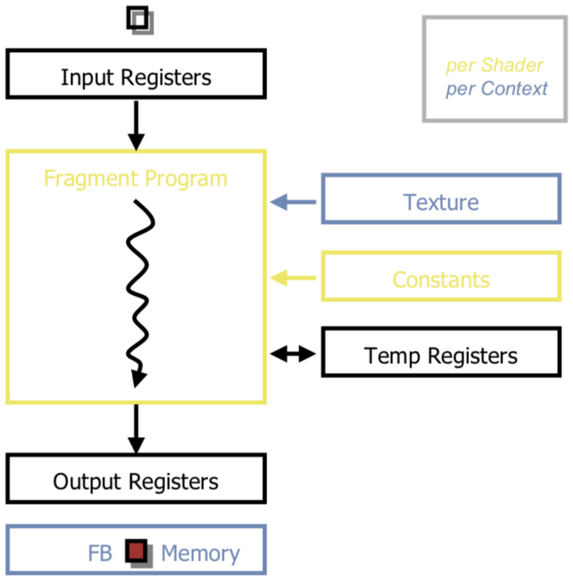
A partnership between NCSA and academic departments.

History of OpenCL. What is (Historical) GPGPU ?

- General Purpose computation using GPU and graphics API in applications other than 3D graphics
  - GPU accelerates critical path of application
- Data parallel algorithms leverage GPU attributes
  - Large data arrays, streaming throughput
  - Fine-grain SIMD parallelism
  - Low-latency floating point (FP) computation
- Applications – see //GPGPU.org
  - Game effects (FX) physics, image processing
  - Physical modeling, computational engineering, matrix algebra, convolution, correlation, sorting

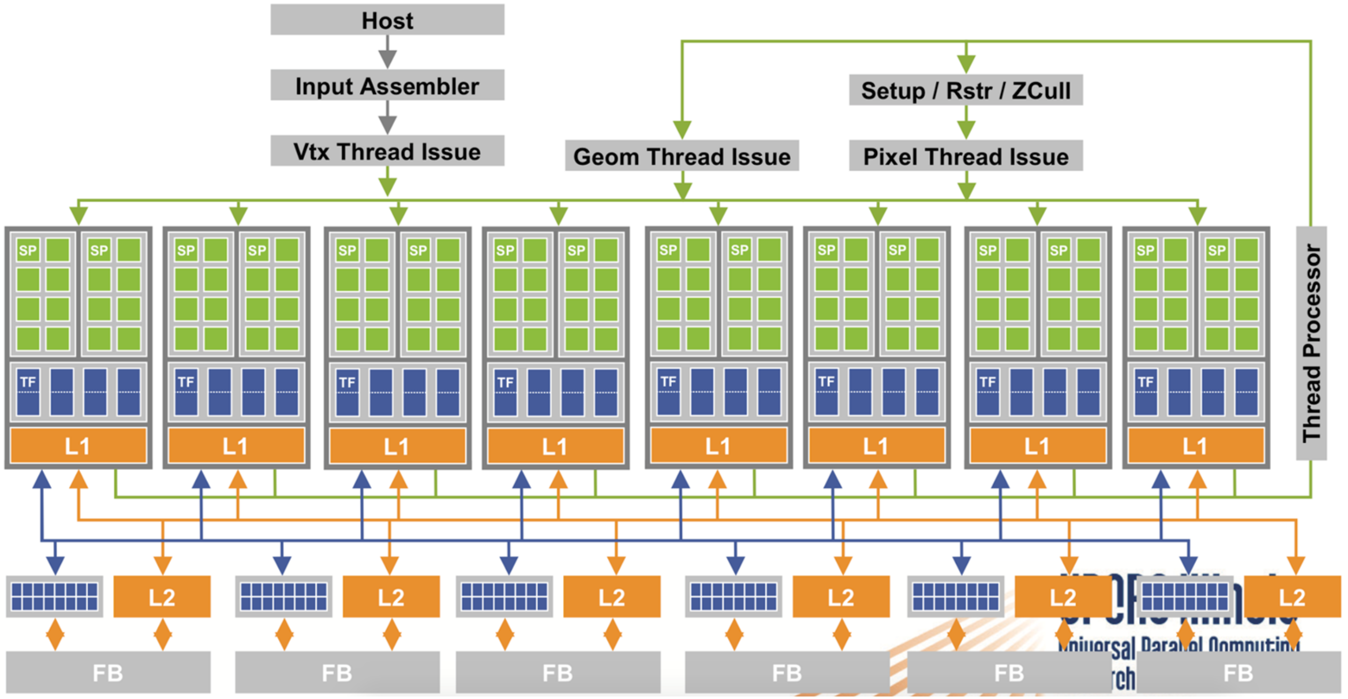
History of OpenCL. Previous GPGPU Constraints

- Dealing with graphics API
  - Working with the corner cases of the graphics API per thread
- Addressing modes
  - Limited texture size/dimension Constants
- Shader capabilities
  - Limited outputs Output Registers
- Instruction sets
  - Lack of Integer & bit ops FB Memory
- Communication limited
  - Between pixels
  - Scatter a[i] = p



History of OpenCL. G80 – Graphics Mode

- The future of GPUs is programmable processing
- So – build the architecture around the processor

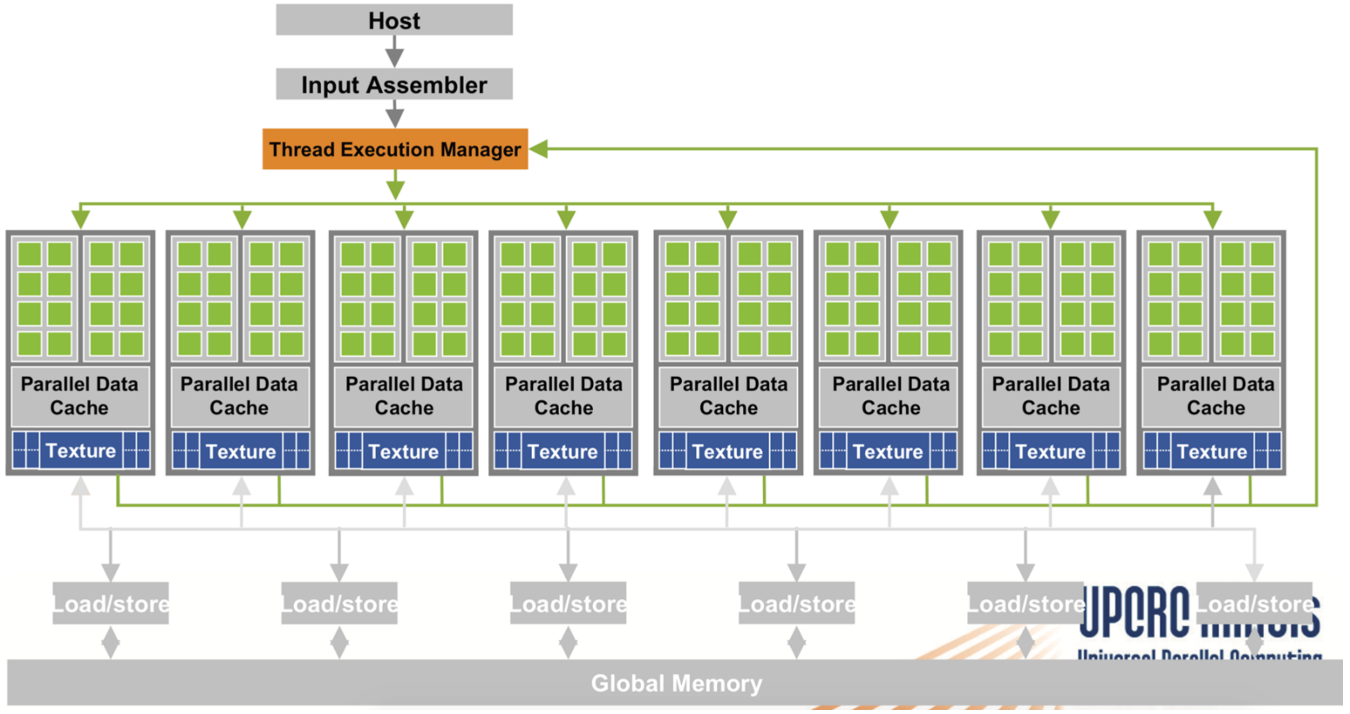


History of OpenCL. CUDA – Recent OpenCL Predecessor

- “Compute Unified Device Architecture”
- General purpose programming model
  - User kicks off batches of threads on the GPU
  - GPU = dedicated super-threaded, massively data parallel co-processor
- Targeted software stack
  - Compute oriented drivers, language, and tools
- Driver for loading computation programs into GPU
  - Standalone Driver - Optimized for computation
  - Interface designed for compute – graphics-free API
  - Data sharing with OpenGL buffer objects
  - Guaranteed maximum download & readback speeds
  - Explicit GPU memory management

History of OpenCL. G80 CUDA mode – A Device Example

- Processors execute computing threads
- New operating mode/HW interface for computing

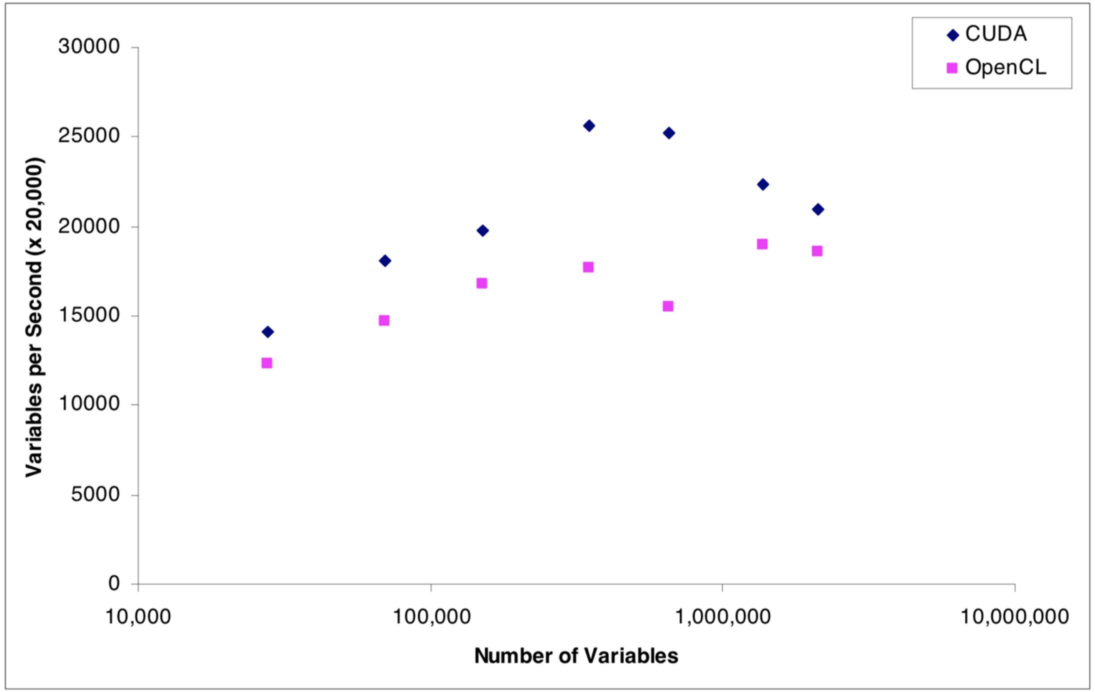


What is it?

- Cross-platform parallel computing API and C-like language for heterogeneous computing devices
- Code is portable across various target devices:
  - Correctness is guaranteed
  - Performance of a given kernel is not guaranteed across differing target devices
- OpenCL implementations already exist for AMD and NVIDIA GPUs, x86 CPUs
- In principle, OpenCL could also target DSPs,Cell, and perhaps also FPGAs

Problems with OpenCL

- **poor performance** compared to CUDA, but **with each new driver release**, the performance of OpenCL under CUDA is **getting closer** to the performance of **CUDA applications**



Processing speed for different problem sizes

Brief description

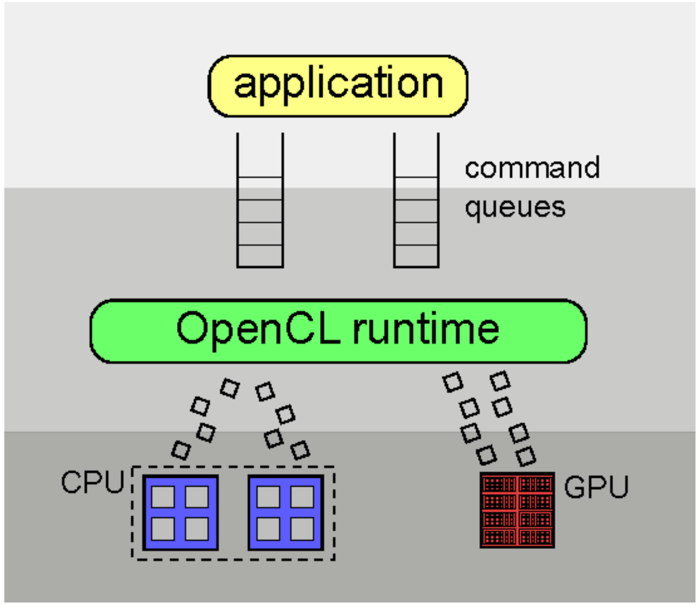
- the OpenCL model turned out to be **very universal**, while it remains **low-level**, allowing you to **optimize applications** for a **specific architecture**.
- It also **provides cross-platform** when moving from one type of OpenCL-devices to another.
- The provider of the OpenCL implementation has the ability to **optimize the interaction** of its device with the OpenCL API in every possible way, seeking to increase the efficiency of resource allocation for the device.
- In addition, a correctly written OpenCL application will remain **effective over generations** of devices.

To sum up

OpenCL technology is of interest to **various IT companies** - from game developers to chip manufacturers.

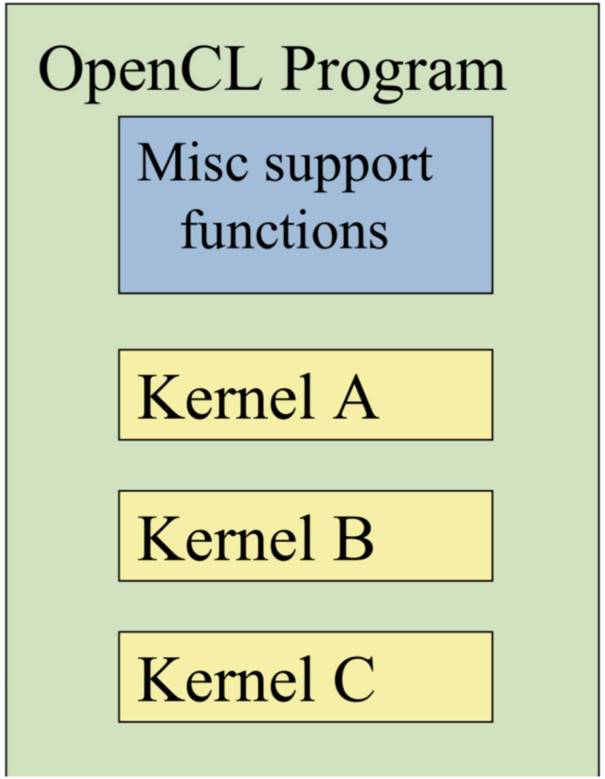
OpenCL was conceived as a technology for **creating applications** that could **run in a heterogeneous environment**.

Moreover, it is designed to **provide comfortable operation** with such **devices** that are **now only in the plans** and even with those that **no one has yet invented**.



OpenCL Programs

- An OpenCL “program” contains one or more “kernels” and any supporting routines that run on a target device
- An OpenCL kernel is the basic unit of parallel code that can be executed on a target device

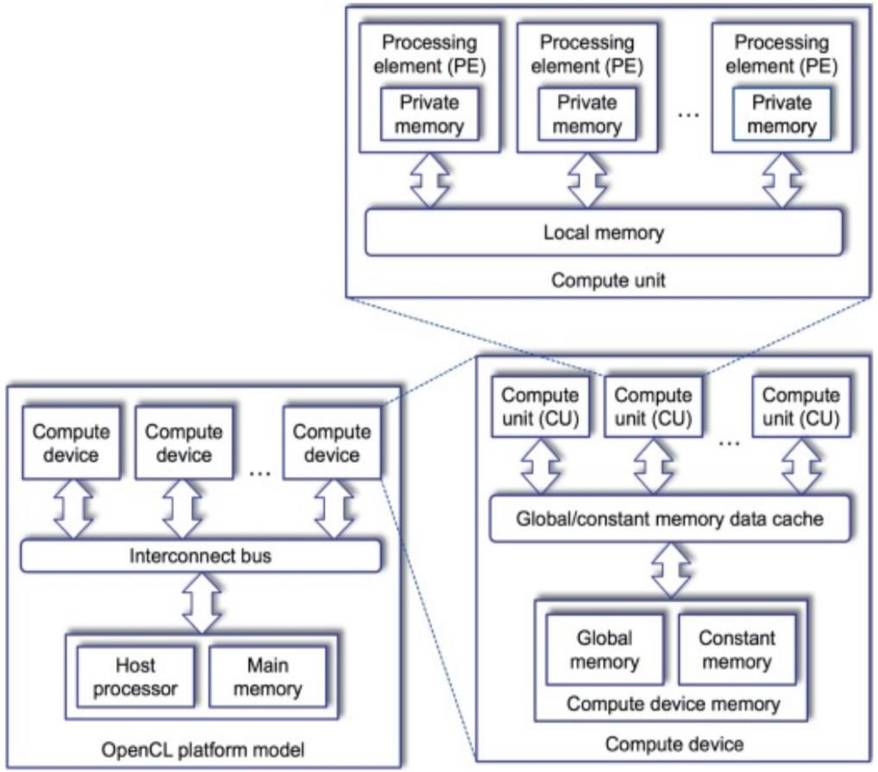


More information

To **coordinate** the work of all these devices, a heterogeneous system always has **one “main” device** that interacts with all other means of the OpenCL API. Such a device is called a "**host**", it is **defined outside** of OpenCL.

This device is supposed to be used for **calculations**, it has a certain “processor” in the general sense of the word. Something that can execute commands. OpenCL is designed for **parallel computing**, such a processor may have means of parallelism within itself.

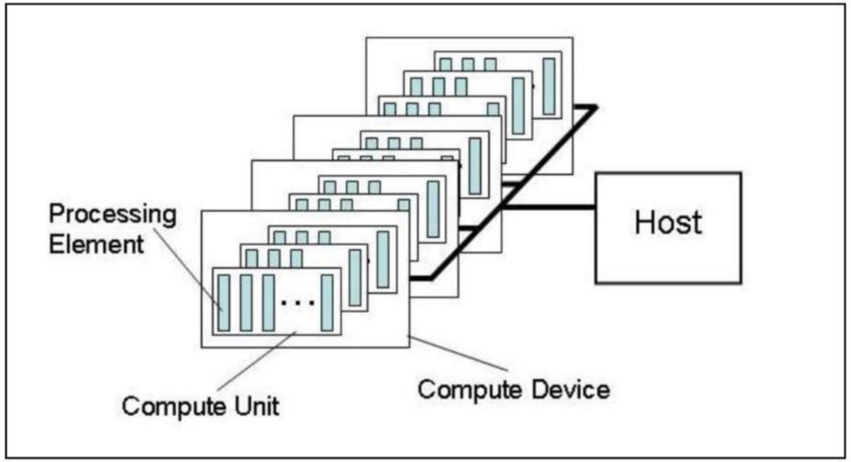
In addition to computing resources, the device has a certain amount of memory.



Platform Model

One Host + one or more Compute Devices

- The OpenCL platform consists of a host connected to devices that support OpenCL.
- Each OpenCL device consists of Compute Unit, which are further divided into one or more processing elements (Processing Elements, hereinafter referred to as PE).



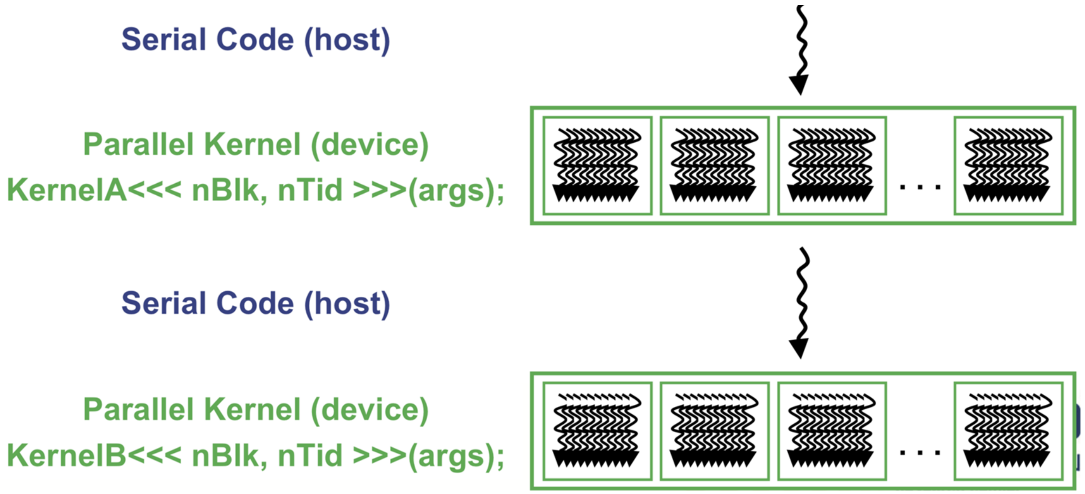
The application is not rigidly connected with OpenCL, which means you can always replace the OpenCL implementation without disrupting the program’s performance

Execution Model

Running an OpenCL Program

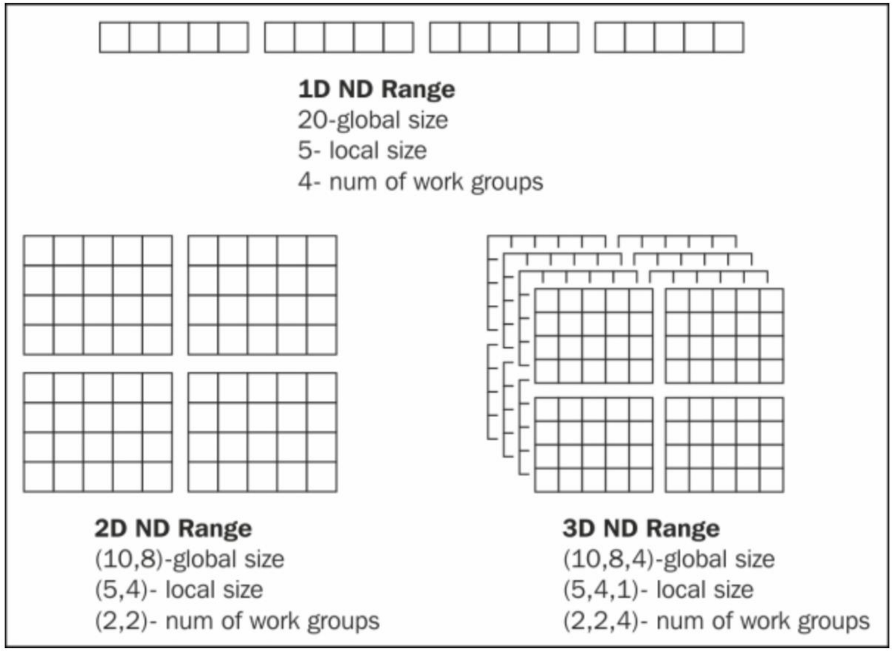
host part of the program

kernels



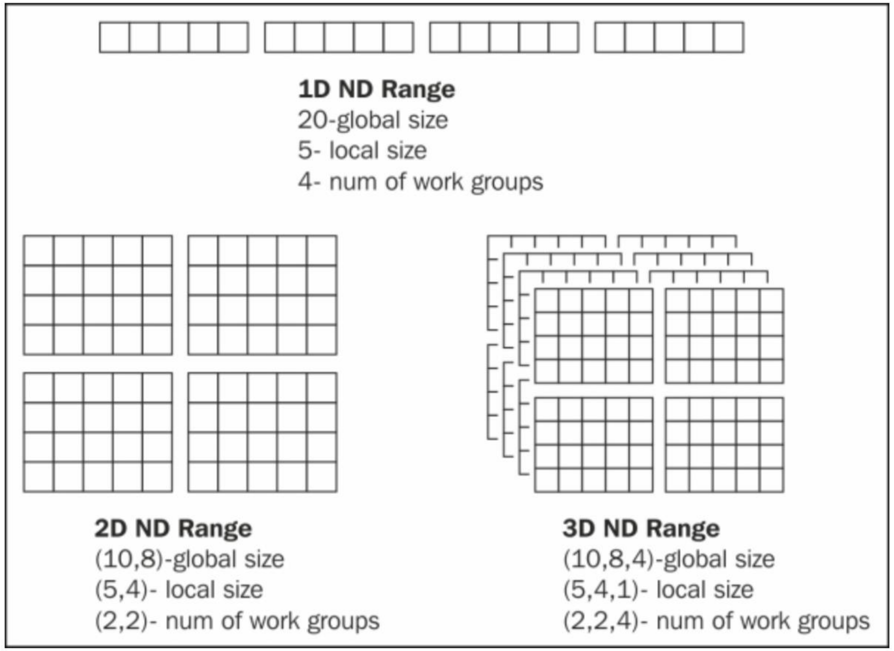
Execution Model

The main purpose of the host program is to create and query the platform and device attributes, define a context for the kernels, build the kernel, and manage the execution of these kernels.



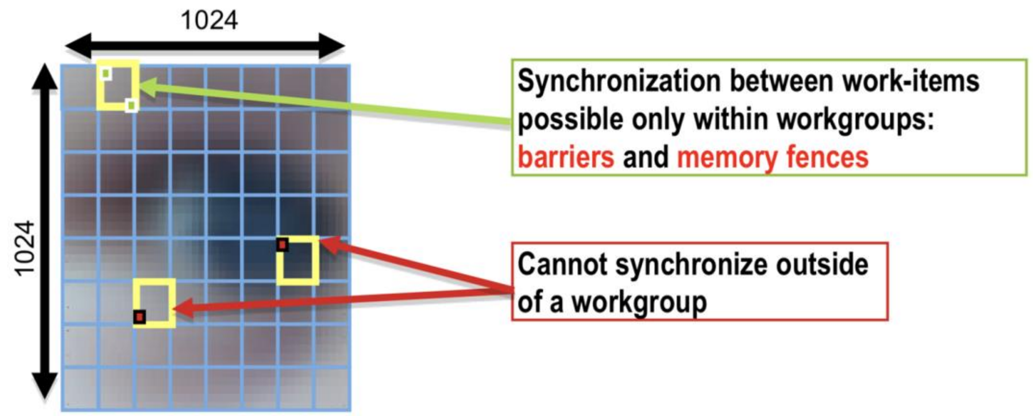
Execution Model

On submission of the kernel by the host to the device, an N dimensional index space is created. N is at least 1 and not greater than 3. Each kernel instance is created at each of the coordinates of this index space. This instance is called as the "work item" and the index space is called as the NDRange. In the following screenshot we have shown the three scenarios for 1, 2 and 3 dimensional NDRange



An N-dimension domain of work-items

- Kernels executed across a global domain of work-items
- Work-items grouped into local workgroups
- Define the “best” N-dimensioned index space for your algorithm
  - Global Dimensions: 1024 x 1024 (whole problem space)
  - Local Dimensions: 128 x 128 (work group … executes together)

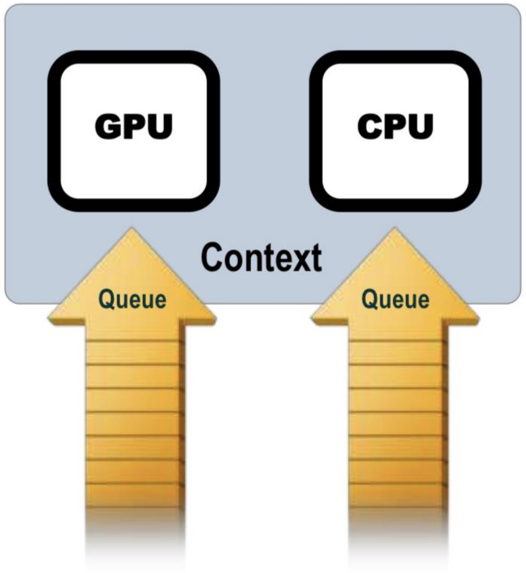

1024
1024
Synchronization between work-items possible only within workgroups: barriers and memory fences
Cannot synchronize outside of a workgroup

Execution context and command queues in the OpenCL execution model.

The host determines the execution context of the kernel.

The context includes the following resources:

- **Devices**: A set of OpenCL devices that the host uses.
- **Kernels**: OpenCL functions that run on devices.
- **Program Objects**: source codes and kernel's executable files.
- **Memory Objects**: A collection of objects in memory that are visible to both the host and the OpenCL device. Memory objects contain values that kernels can work with.

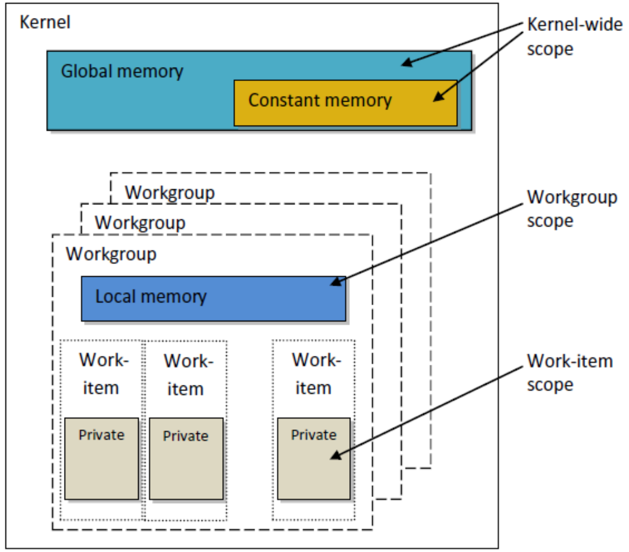


Execution context and command queues in the OpenCL execution model.

The **context** is created and managed by means of functions **from the OpenCL API**. The **host creates** a **data structure** called a command-queue to control the **execution of the kernel's devices**. The **host sends** the **commands to the queue**, after which they are set by the **scheduler** for execution on **devices in the desired context**.

Commands can be of the following types:

- **Kernel execution command**: execute the kernel on a PEs device.
- **Memory Commands**: Move data to, from, or between memory objects.
- **Synchronization Commands**: Controls the execution order of commands.

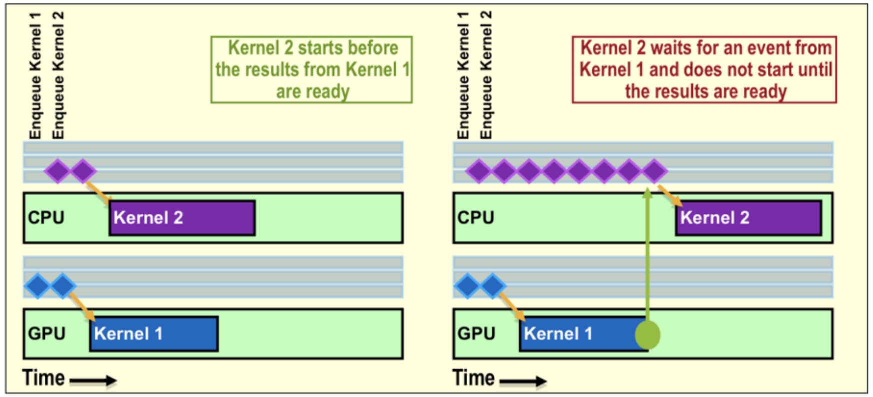


Execution context and command queues in the OpenCL execution model.

The command queue schedules commands for execution on the device. **They execute asynchronously between the host and device**.

Commands can be executed relative to each other in two ways:

- **Execution in order**
- **Inconsistent execution**


Enqueue Kernel 1
Enqueue Kernel 2
Kernel 2 starts before the results from Kernel 1 are ready
CPU
Kernel 2
GPU
Kernel 1
Time
Enqueue Kernel 1
Enqueue Kernel 2
Kernel 2 waits for an event from Kernel 1 and does not start until the results are ready
CPU
Kernel 2
GPU
Kernel 1
Time

Execution context and command queues in the OpenCL execution model.

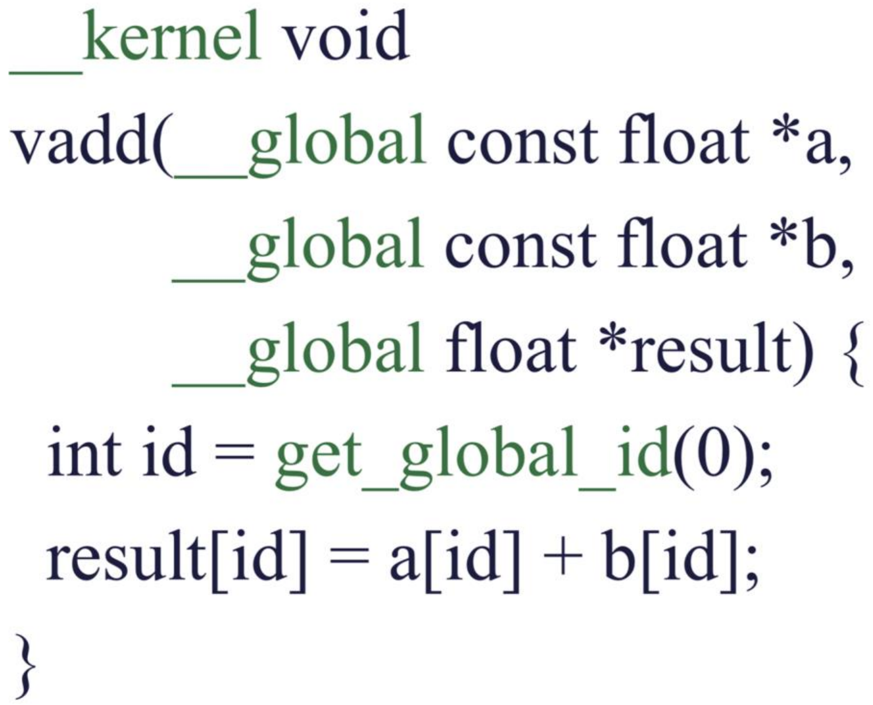
Using the **command queue** allows for great **versatility and flexibility** when using OpenCL.

Modern GPUs have their own scheduler, which decides what to execute and when and on which computing units.

Using the **queue does not hamper the work of the scheduler**, which has its own queue of commands.

OpenCL Kernels

- Code that actually executes on target devices
- Kernel body is instantiated once for each work
  - An OpenCL work item is equivalent to a CUDA
- Each OpenCL work item gets a unique index



```
__kernel void
vadd(__global const float *a,
     __global const float *b,
     __global float *result) {
  int id = get_global_id(0);
  result[id] = a[id] + b[id];
}
```

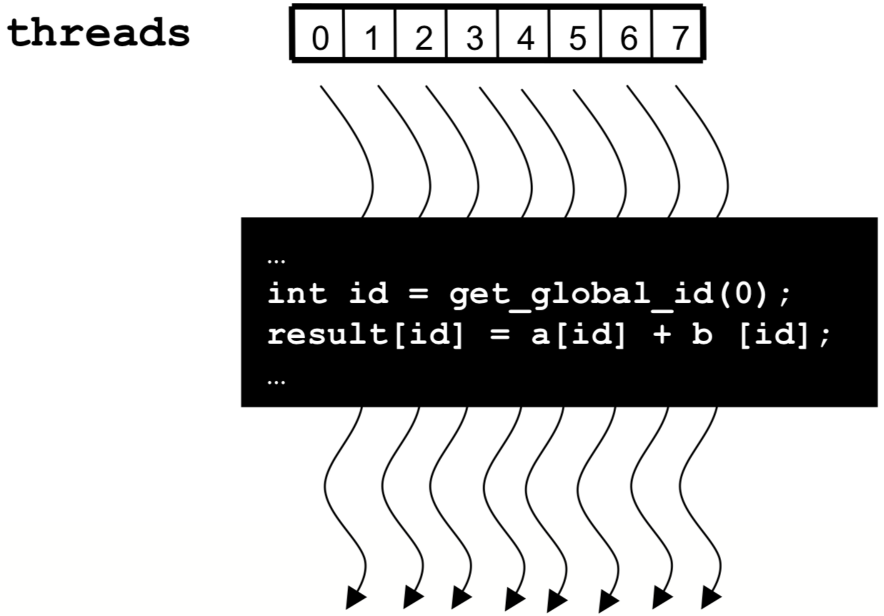
Execution Model: kernel categories

- **OpenCL kernel**: written in OpenCL C and compiled by the OpenCL compiler. All OpenCL implementations must support OpenCl-kernel.
- **Naitive kernel**: access to them is through host pointers to a function. Native kernel is queued for execution, as well as OpenCL-kernel and uses the same memory objects as OpenCL-kernel. For example, such kernels can be functions defined in the application code or exported from the library.

Array of Parallel Work Items

An OpenCL kernel is executed by an array of work items

- All work items run the same code (SPMD)
- Each work item has an index that it uses to compute memory addresses and make control decisions

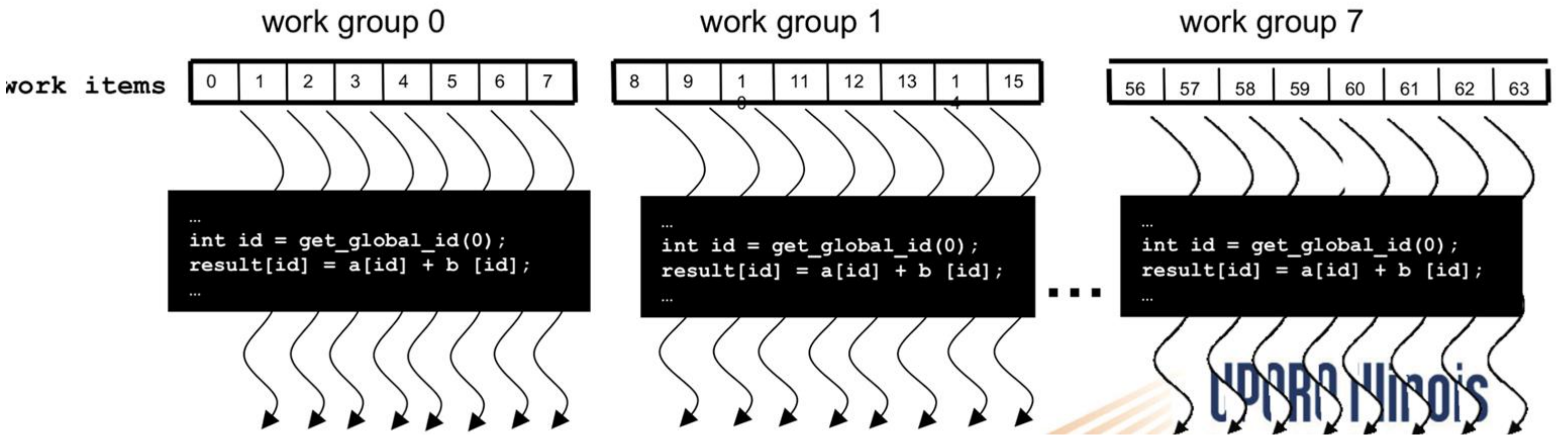


```
…
int id = get_global_id(0);
result[id] = a[id] + b [id];
…
```

Work Groups: Scalable Cooperation

Divide monolithic work item array into work groups

- Work items within a work group cooperate via shared memory, atomic operations and barrier synchronization
- Work items in different work groups cannot cooperate



```
…
int id = get_global_id(0);
result[id] = a[id] + b [id];
…
```

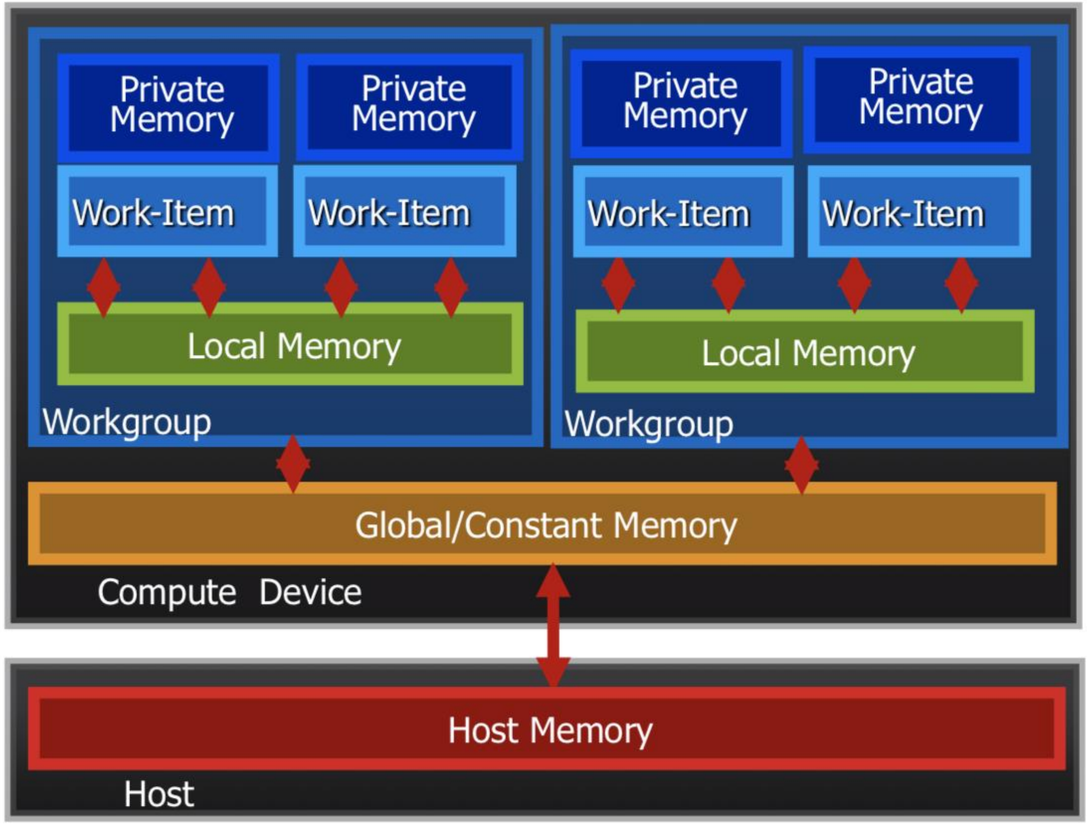
Memory Model

- **Global memory.** This memory provides read and write access to elements of all groups.
- **Constant memory.** A region of global memory that remains constant during kernel execution.
- **Local memory.** A region of memory local to the group.
- **Private (private) memory.** A memory area owned by a Work-Item.

The specification defines 4 types of memory, but again **does not impose** any **requirements** on the **implementation of memory** in hardware.

The existence of precisely these types of memory is **quite logical**: the processor core has its own cache, the processor has a common cache and the entire device has a certain amount of memory.

Memory Model



Memory management is Explicit:

You must move data from host -> global -> local … and back

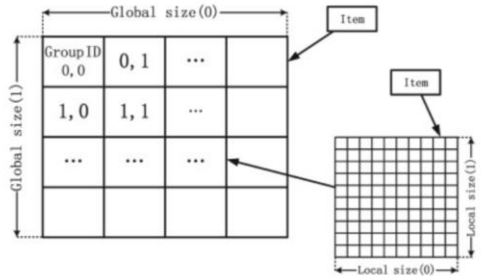
Programming Model

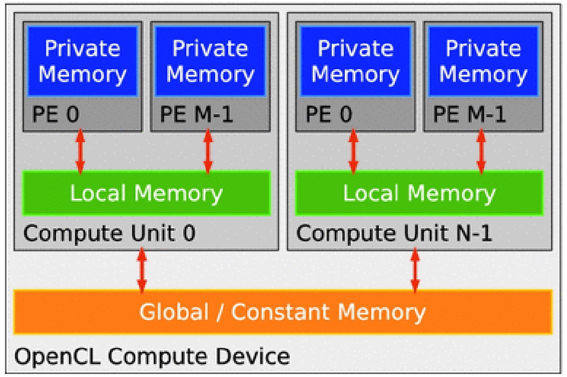
OpenCL Execution Model

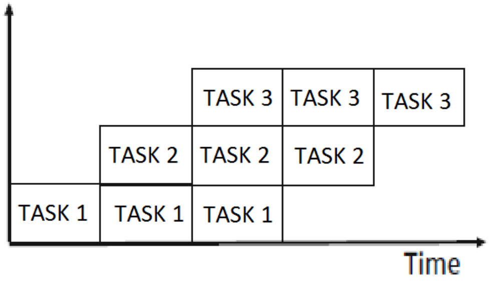
Data Parallel

hybrid models

Task Parallel







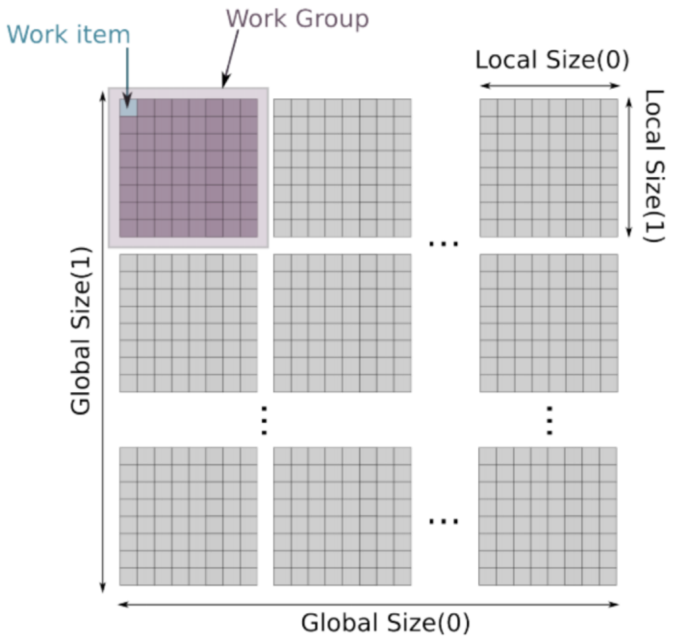
A software model with data parallelism.

This model defines computation as a sequence of instructions applied to many elements of a memory object.

OpenCL provides a hierarchical model of data concurrency.

In an **explicit mode**l, the programmer determines the total number of elements that must be executed in parallel and **how these elements** will be **distributed into groups**.

In an **implicit model**, the programmer only determines the total number of elements that must be executed in parallel, and **the division into work groups** is performed automatically.

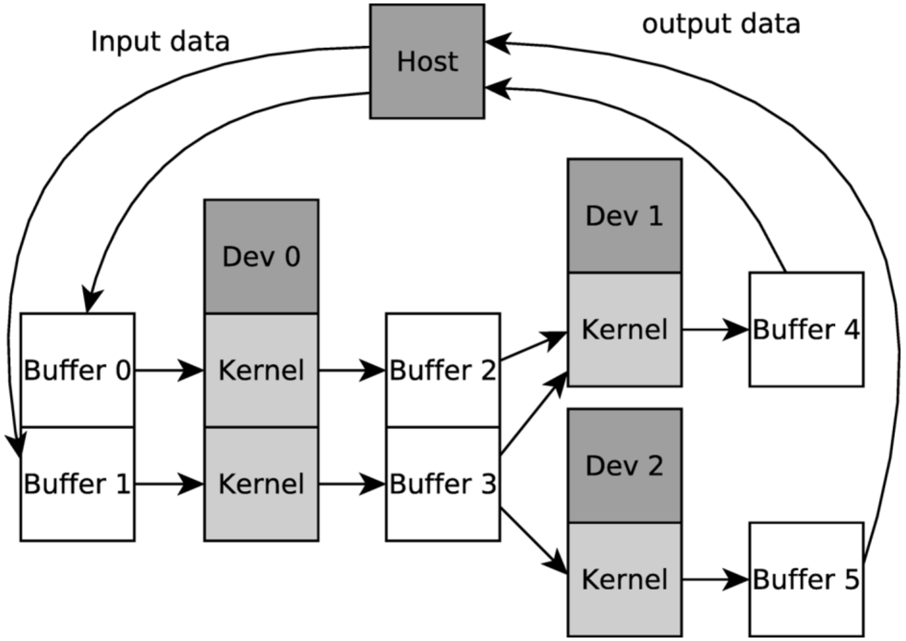


Software model with job parallelism.

In this model, each copy of the kernel is executed regardless of any index space.

In this model, users express concurrency in the following ways:

- use vector data types implemented in the device;
- queue a lot of tasks;
- queue native kernels using a software model orthogonal to OpenCL;



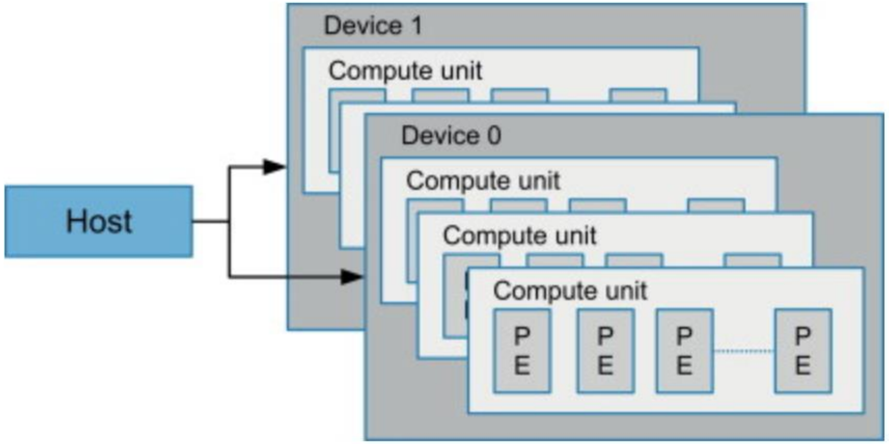
Which model is better?

For modern GPUs and Cells, the **first model** is well suited. **But not all algorithms** can be effectively **implemented within the framework** of such a model, and there is also the likelihood of a device appearing whose **architecture will be inconvenient** for using the first model.

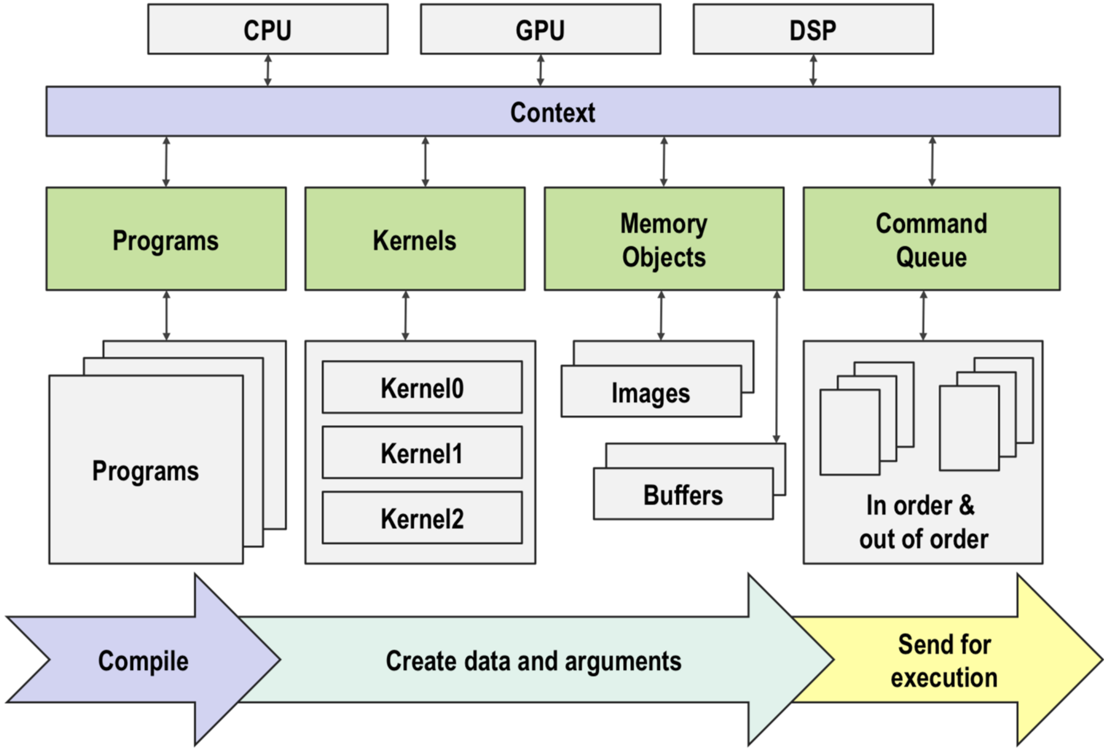
In this case, the **second model** allows you to write **applications specific** to another architecture.

What does the OpenCL platform consist of?

- **OpenCL Platform Layer:** Allows the host to discover OpenCL devices, query their properties, and create context.
- **OpenCL Runtime:** The runtime allows a host program to manage contexts after they have been created.
- **OpenCL Compiler:** The OpenCL compiler creates executable files containing the OpenCL – kernel.



Creating an OpenCL Program



Introduction to Java and OpenCL Development

OpenCL is an open standard and there are several implementations from different manufacturers of hardware AMD, Intel, Nvidia, IBM, etc. for the operating systems Windows, Linux and MacOS.

When host code is part of the OpenCL code that prepares data and coordinates the work of OpenCL kernels, it is written in Java, there are problems with debugging in Linux / Windows.

Introduction to Java and OpenCL Development

Recall the success of the Java platform in the market for developing cross-platform server software, **it is not clear why hardware vendors** still do **not develop OpenCL development tools** - debugging and profiling built into the Eclipse IDE, Netbeans IDE, IntelliJ Idea.

In the meantime, these development environments **do not include ready-made tools** that simplify working with the technology in question.

Benefits of sharing Java and OpenCL

- Cross-platform
- A huge number of developers / architects who know this platform.
- A large number of open source libraries are available for use in development.
- Platform for cloud applications
- Recognized corporate standard in banks, financial institutions, telecommunications
- Many programming languages that can be run in the JVM: jruby, PHP (Caucho quercus), jython, javascript (Mozilla Rhino), Scala, etc.
- JavaCL is a compact and object-oriented API for OpenCL host code.

Drawbacks of sharing Java and OpenCL

- Non-dereferencing of the garbage collector. Your application may require low latency in processing data and generating a response
- The presence of large overhead costs for copying data between the JVM and native code, including implementations of OpenCL. This also includes the lack of code for jvm that can read and write the data you need
- The algorithm cannot be effectively implemented within the framework of the OpenCL architecture. Not all algorithms are well parallelized (Amdahl-Ware law), the data transfer time on the bus can be many times longer than the computation time on the device, etc.

To sum up about using Java and OpenCL

Due to the prospects for the development of computing devices along the path of increasing parallelism, this approach to software development claims to become one of the main in the development of high-performance enterprise and cloud applications on the JVM.

Preparation

We need:

1. specification of the standard - https://www.khronos.org/registry/OpenCL
2. SDK (AMD or NVidia)
3. OpenCL literature - https://developer.nvidia.com/opencl

If you install the Nvidia Computing SDK, you will automatically receive all the necessary documents.

Preparation

In addition, as a bonus, you will receive many interesting examples of programs (30 pieces in the latest SDK release). Thanks to these examples, it’s easy to learn how to use OpenCL correctly.

The OpenCL compiler is built into the driver, so the choice of IDE for development is not limited in any way, therefore I will not describe the process of setting up any specific IDE. All you have to do is write the paths to the headers and libraries that the SDK will install.

Bibliography

- https://imagej.net/A_Tutorial_for_using_OpenCL_in_ImageJ
- https://habr.com/ru/post/73526/
- https://developer.nvidia.com/opencl
- http://developer.amd.com/GPU/ATISTREAMSDKBETAPROGRAM/Pages/default.aspx
- https://www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingGuide.pdf
- https://www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_OpenCL_BestPracticesGuide.pdf
- http://forums.amd.com/devforum/categories.cfm?catid=390&entercat=y

Bibliography

- https://www.khronos.org/registry/OpenCL/sdk/1.1/docs/man/xhtml/
- https://habr.com/ru/post/72247/
- https://habr.com/ru/post/72650/
- https://habr.com/ru/post/124873/
- https://www.researchgate.net/publication/273379680_Tutorial_para_OpenCL_en_ANSI_C_Java_y_C
- https://askvoprosy.com/voprosy/opencl-and-java
- https://habr.com/ru/post/261323/
- https://habr.com/ru/post/134500/

Bibliography

- https://subscription.packtpub.com/book/application_development/9781849692342/2/ch02lvl1sec17/execution-model
- https://www.khronos.org/assets/uploads/developers/library/2012-pan-pacific-road-show-June/OpenCL-Details-Taiwan_June-2012.pdf
- https://www.ks.uiuc.edu/Research/gpu/files/upcrc_opencl_lec1.pdf
- https://www.sciencedirect.com/topics/computer-science/opencl-platform
- http://www.ncsa.uiuc.edu/Projects/GPUcluster/
- https://arxiv.org/vc/arxiv/papers/1005/1005.2581v1.pdf
- http://multicore.doc.ic.ac.uk/opencl/opencl-2.0.pdf