University of Applied Sciences Koblenz RheinAhrCampus
Faculty of Mathematics and Engineering
Module: Parallel Computing

Instructor: Prof. Schmidt/ Prof. Jaekel
Contributer: Jan Beneke
WS 2011/12
16.01.2012

OpenCL

A brief introduction for NVIDIA CUDA programmers

Open Computing Language

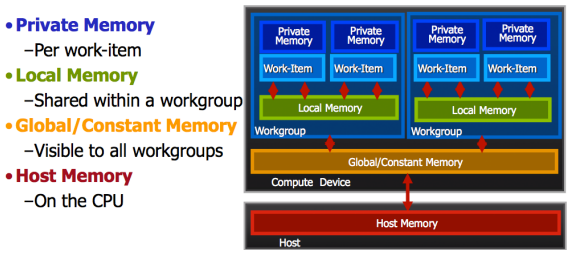
A language extension (mainly C) for accessing heterogeneous computational devices (CPUs, GPUs, accelerator cards).
- Parallel code execution on single or multiple devices of many vendors (e.g. NVIDIA, AMD/ ATI, Intel, IBM, ARM, ...).
- Vender neutral and royalty free.
- Specifications under review by Khronos OpenCL working group - developed with industry leaders.
- Profiles for desktop machines, server farms (MPI-integration) and even hand-held devices.
- Interoperable with OpenGL and Microsoft Direct 3D graphics APIs.

Design goals

An easy way to allow, with sparse new syntax and extensions, parallel computing (data- or task-parallel) with all available devices in a modern system and without writing new code for every type of device - or short:
One portable program uses all available resources.

Abstract hardware model

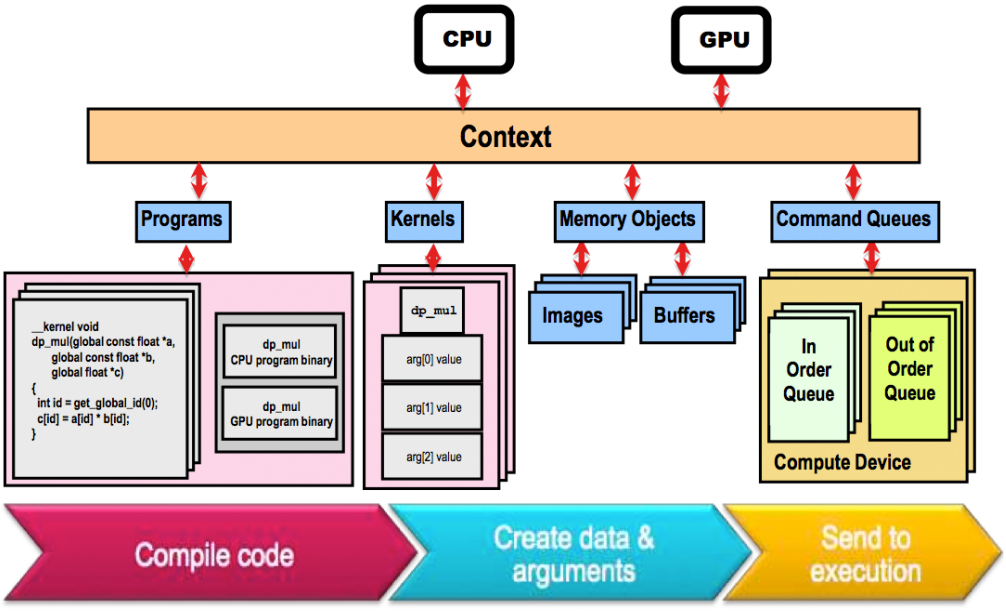
- *Platform model*: One host system controls multiple computing *devices* (e.g. CPUs or GPUs). Each device contains computing units (*workgroups*) consisting of processing elements (*work-items*).
- *Memory model*: An abstracted shared memory model (like in CUDA), that is independent of the based hardware.



Execution model

The application runs on the host and submits work to the device(s). Basic elements needed to execute OpenCL code[1]
- *Work-item*: Basic unit on a OpenCL device (e.g. one ALU/ core).
- *Command queue*: Queues Kernel executions and synchronizes if needed (in-order[2] or out-of-order[3]) - event handling.
- *Kernel*: The code to be executed on the devices - extended C function.
- *Program*: Collection of kernels and other functions - analogous to a dynamic library. It has to be compiled for every device architecture at runtime.
- *Context*: Environment to execute the work item in, includes devices, memories and command queues.

Workflow overview

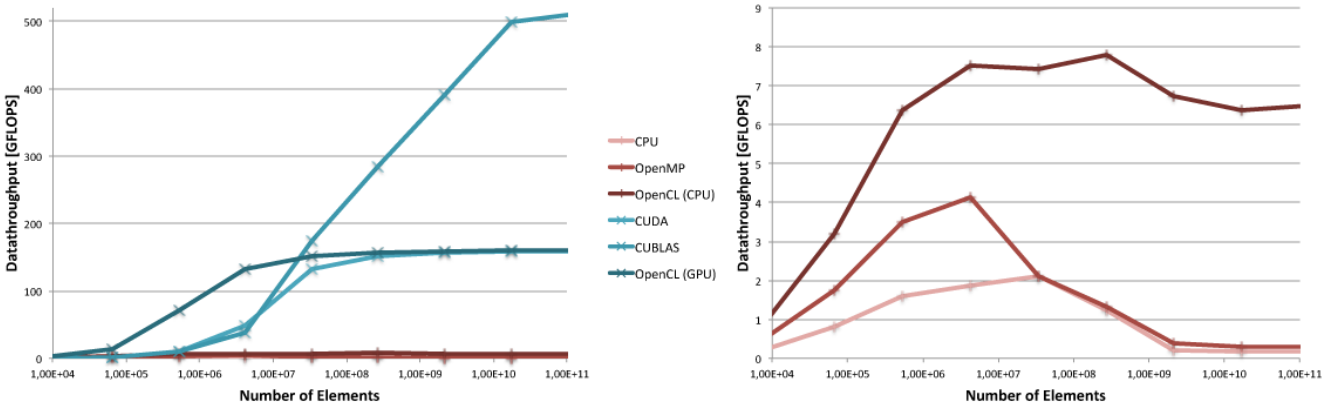


[1] All elements but the work-items must be created by the programmer in the host application.
[2] In-order queues: One queue per device.
[3] Out-of-order queues: Multiple queues per device

Benchmark

System: *CPU*: Intel Core i3 - 2 Cores (2 Threads each) @ 3.1 GHz - 3MB cache
RAM: Corsair - 8GB DDR3 (1333 MHz)
GPU: MSI - NVIDIA GeForce GTX 5600Ti - 384 cores @ 880MHz - 1024MB GGDR5 (4200MHz)
Software: *Compiler*: Microsoft Visual Studio 2008 Professional x86
OS: Microsoft Windows 7 Professional SP1 x64
APIs: OpenCL 1.2[4], NVIDIA CUDA 4.0[4], NVIDIA CUBLAS 2.0[4], OpenMP 2.0[5]
Problem: $A^{2048\times8192} \cdot B^{4096\times2048} = C^{4096\times8196} \Rightarrow 1.374 \cdot 10^{11}$ elements to compute
Results:

		Execution time $[s]$	Throughput $[GFLOPS/s]$
CPU	Single threaded	818.3	0.16
	OpenMP	460.05	0.30
	OpenCL 1.2	**21.76**	**6.32**
GPU	NVIDIA CUDA 4.0	0.87	158.23
	OpenCL 1.2	**0.86**	**159.72**
	NVIDIA CUBLAS 2.0	0.24	580.90



Conclusion: *CPU*: OpenCL executes the code not on every core available but on every possible thread on a core (e.g. OpenMP doesn't). In addition OpenCL uses a very advanced event handling to queue the kernel execution.

GPU: OpenCL offers approximately the same computation speed as other GPGPU libraries such as NVIDIA CUDA or ATI Stream. The disadvantage to other libraries lies in the more complex workflow that has to be maintained and in the lag of delivered libraries for common tasks like CUBLAS as a fast BLAS[6] implementation for NVIDIA CUDA.

Summary: OpenCL vs. NVIDIA CUDA

	OpenCL 1.2	**NVIDIA CUDA 4.0**
Free of charge	✓	✓
Vendor independent/ Multiple SDK-vendors	✓	
Open standard	✓	
Use different types of heterogeneous hardware	✓	
Cross-platform	✓	✓
Interoperability with graphic libraries	✓	✓
Included libraries for common tasks		✓
Peer-to-peer-communication between devices		✓
Global virtual addressing space		✓

Conclusion: OpenCL is slightly more complicated to code (see workflow) than NVIDIA CUDA but is not limited to using GPUs and can use devices from multiple vendors. NVIDIA CUDA is more enhanced and offers more features and well-engineered libraries such as CUBLAS.

References:

Khrohnos Group: `http://khronos.org/opencl/`
NVIDIA: `http://developer.nvidia.com/opencl/`
Intel: `http://software.intel.com/en-us/articles/vcsource-tools-opencl-sdk/`
AMD/ ATI: `http://developer.amd.com/zones/OpenCLZone/`

Online resources:

Further information as well as the benchmark and example source code is available at `http://www.janbeneke.de/ParComp` .

[4]Part of the *NVIDIA GPU Computing SDK 4.0*.
[5]Part of *Microsoft Visual Studio 2008 Professional x86*.
[6]Basic Linear Algebra Subprograms: A de facto API standard for basic linear algebra operations.