GPU Programming using OpenCL

Blaise Tine
School of Electrical and Computer Engineering
Georgia Institute of Technology

Outline
- What’s OpenCL?
- The OpenCL Ecosystem
- OpenCL Programming Model
- OpenCL vs CUDA
- OpenCL VertorAdd Sample
- Compiling OpenCL Programs
- Optimizing OpenCL Programs
- Debugging OpenCL Programs
- The SPIR Portable IL
- Other Compute APIs (DirectX, C++ AMP, SyCL)
- Resources
What’s OpenCL?

- Low-level programming API for data parallel computation
  - Platform API: device query and pipeline setup
  - Runtime API: resources management + execution
- Cross-platform API
  - Windows, MAC, Linux, Mobile, Web…
- Portable device targets
  - CPUs, GPUs, FPGAs, DSPs, etc…
- Implementation based on C99
- Maintained by Kronos Group (www.khronos.org)
- Current version: 2.2 with C++ support (classes & templates)

OpenCL Implementations
OpenCL Front-End APIs

OpenCL Platform Model

- Multiple compute devices attached to a host processor
- Each compute device has multiple compute units
- Each compute unit has multiple processing elements
- Each processing element execute the same work-item within a compute unit in log steps.
OpenCL Execution Model

- A **kernel** is a logical unit of instructions to be executed on a compute device.
- Kernels are executed in multi-dimensional index space: **NDRange**
- For every element of the index space a **work-item** is executed
- The index space is tiled into **work-groups**
- Work items within a workgroup are synchronized using barriers or fences

OpenCL Memory Model

- **Global Memory**
  - Shared memory accessible to all work-items + host
- **Constant/texture Memory**
  - Read-only shared memory accessible to all work-items + host
- **Local Memory**
  - Sharing data between work-items within same work-group
- **Private Memory**
  - Only accessible within work-item
  - Implemented as Register File
OpenCL vs CUDA

- OpenCL terminology aims for generality

<table>
<thead>
<tr>
<th>OpenCL Terminology</th>
<th>CUDA Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Unit</td>
<td>Streaming Processor (SM)</td>
</tr>
<tr>
<td>Processing Element</td>
<td>Processor Core</td>
</tr>
<tr>
<td>Wavefront (AMD)</td>
<td>Warp</td>
</tr>
<tr>
<td>Work-item</td>
<td>Thread</td>
</tr>
<tr>
<td>Work-group</td>
<td>Thread Block</td>
</tr>
<tr>
<td>NDRange</td>
<td>Grid</td>
</tr>
<tr>
<td>Global Memory</td>
<td>Global Memory</td>
</tr>
<tr>
<td>Constant Memory</td>
<td>Constant Memory</td>
</tr>
<tr>
<td>Local Memory</td>
<td>Shared Memory</td>
</tr>
<tr>
<td>Private Memory</td>
<td>Local Memory</td>
</tr>
</tbody>
</table>

(9)

OpenCL vs CUDA (2)

- Resources Qualifiers

<table>
<thead>
<tr>
<th>Description</th>
<th>OpenCL Terminology</th>
<th>CUDA Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel global function</td>
<td>__kernel</td>
<td><strong>global</strong></td>
</tr>
<tr>
<td>Kernel local function</td>
<td>nothing*</td>
<td><strong>device</strong></td>
</tr>
<tr>
<td>Readonly memory</td>
<td>__constant</td>
<td><strong>device</strong></td>
</tr>
<tr>
<td>Global memory</td>
<td>__global</td>
<td><strong>device</strong></td>
</tr>
<tr>
<td>Private memory</td>
<td>__local</td>
<td><strong>shared</strong></td>
</tr>
</tbody>
</table>

(10)
### OpenCL vs CUDA (3)

**Work Items Indexing**

<table>
<thead>
<tr>
<th>OpenCL Terminology</th>
<th>CUDA Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_num_groups()</code></td>
<td><code>gridDim</code></td>
</tr>
<tr>
<td><code>get_local_size()</code></td>
<td><code>blockDim</code></td>
</tr>
<tr>
<td><code>get_group_id()</code></td>
<td><code>blockIdx</code></td>
</tr>
<tr>
<td><code>get_local_id()</code></td>
<td><code>threadIdx</code></td>
</tr>
<tr>
<td><code>get_global_id()</code></td>
<td><code>blockIdx * blockDim + threadIdx</code></td>
</tr>
<tr>
<td><code>get_global_size()</code></td>
<td><code>gridDim * blockDim</code></td>
</tr>
</tbody>
</table>

### OpenCL vs CUDA (4)

**Threads Synchronization**

<table>
<thead>
<tr>
<th>OpenCL Terminology</th>
<th>CUDA Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>barrier()</code></td>
<td><code>_syncthreads()</code></td>
</tr>
<tr>
<td>No direct equivalent*</td>
<td><code>_threadfence()</code></td>
</tr>
<tr>
<td><code>mem_fence()</code></td>
<td><code>_threadfence_block()</code></td>
</tr>
<tr>
<td>No direct equivalent*</td>
<td><code>_threadfence_system()</code></td>
</tr>
<tr>
<td>No direct equivalent*</td>
<td><code>_syncwarp()</code></td>
</tr>
<tr>
<td><code>Read_mem_fence()</code></td>
<td>No direct equivalent*</td>
</tr>
<tr>
<td><code>Write_mem_fence()</code></td>
<td>No direct equivalent*</td>
</tr>
</tbody>
</table>
OpenCL vs CUDA (5)

- API Terminology

<table>
<thead>
<tr>
<th>OpenCL Terminology</th>
<th>CUDA Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>clGetContextInfo()</td>
<td>cuDeviceGet()</td>
</tr>
<tr>
<td>clCreateCommandQueue()</td>
<td>No direct equivalent*</td>
</tr>
<tr>
<td>clBuildProgram()</td>
<td>No direct equivalent*</td>
</tr>
<tr>
<td>clCreateKernel()</td>
<td>No direct equivalent*</td>
</tr>
<tr>
<td>clCreateBuffer()</td>
<td>cuMemAlloc()</td>
</tr>
<tr>
<td>clEnqueueWriteBuffer()</td>
<td>cuMemcpyHtoD()</td>
</tr>
<tr>
<td>clEnqueueReadBuffer()</td>
<td>cuMemcpyDtoH()</td>
</tr>
<tr>
<td>clSetKernelArg()</td>
<td>No direct equivalent*</td>
</tr>
<tr>
<td>clEnqueueNDRangeKernel()</td>
<td>kernel&lt;&lt;&lt;...&gt;&gt;&gt;(())</td>
</tr>
<tr>
<td>clReleaseMemObj()</td>
<td>cuMemFree()</td>
</tr>
</tbody>
</table>

OpenCL vs CUDA (6)

- Which is Best?

<table>
<thead>
<tr>
<th>Strengths</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>CUDA is better on Nvidia cards</td>
</tr>
<tr>
<td>Device Capabilities</td>
<td>CUDA has an edge</td>
</tr>
<tr>
<td>Portability</td>
<td>CUDA is not portable</td>
</tr>
<tr>
<td>Documentation</td>
<td>CUDA has many online resources</td>
</tr>
<tr>
<td>Tools</td>
<td>CUDA has more mature tools</td>
</tr>
<tr>
<td>Language Accessibility</td>
<td>CUDA C++ extension is nice</td>
</tr>
</tbody>
</table>

(13) (14)
OpenCL Program Flow

- Compile kernel programs
  - Offline or Online
- Load kernel objects
- Load application data to memory objects
- Build command queue
  - Batch instructions
  - Defer rendering
- Submit command queue
- Execute program

Compiling OpenCL Programs

- The compiler tool chain uses LLVM optimizer
- LLVM generates a device specific IL for the target GPU
- LLVM can also generate CPU target binary
- CPU target can be used for verification and debugging
OpenCL VertexAdd Sample

kernel qualifier

Global thread index

Vector addition

OpenCL VertexAdd Sample (2)

Setup kernel grid

Allocate host resources

Create device context

Allocate device resources

Populate device memory
OpenCL VertexAdd Sample (3)

Build kernel program

Set kernel arguments

Launch kernel execution

Read destination buffer

```c
// Build the program
program = clCreateProgramWithSource(device, 1, (cl seedu **...

// Create the kernel
kernel = clCreateKernel(program, "VertexAdd", &kernel)

// Set kernel the Arguments
cfSetKernelArg(kernel, 0, sizeof(lo_meed), &lo_meed);

// Launch kernel
clEnqueueNDRangeKernel(commandQueue, kernel, 1, NULL, 4*1024, 4*1024, 4*1024, 0, NULL, NULL);
```

(19)

---

Optimizing OpenCL Programs

- **Profile before optimizing!**

- Fine grain workload partitioning
  - **Subdivide work to feed all compute resources**

- Use constant memory when possible
  - **Implementations may optimized access to the data.**

- Use local memory
  - **Much faster than global memory**
  - **Tile computation to fit local memory**

- Reduce thread synchronization

- Reduce host-device communication overhead
  - **Command Queue batching and synchronization**

---

10
Debugging OpenCL Programs

- Debugging Support is vendor specific
- Most implementations support debugging via GDB
- Compile program with debug symbols
  - `clBuildProgram()` with "-g"
- Internally uses CPU target for efficiency
- Runtime Profiling supported
  - `clGetEventProfilingInfo()`

The SPIR Portable IL

- Portable equivalent of Nvidia PTX

```
clBuildProgram( "-x spir -spir-std=1.0"..., )
```

- SPIR Verifier
- Standard LLVM optimizations
- Custom optimizations
  - E.g. Dead code elimination
- Materialization
  - (Convert to device specific IR)
- ABI fixup, triple, vectorize, custom optimizations
- JIT

© Copyright Khronos Group
Other Compute APIs: DirectX 12

- Full-featured compute API
- Major vendors support
  - Nvidia, AMD, Intel
- Optimized for Gaming
  - Graphics
  - AI
- Windows only

https://www2.cs.duke.edu/courses/compsci344/spring15/classwork/15_shading

Other Compute APIs: C++ AMP, SyCL

- Single source compute API
- Exploit modern C++ lambda extension
- Productivity without performance lost!
Resources

- API specifications:  
  https://www.khronos.org/registry/OpenCL

- Open-source implementation:  
  https://01.org/beignet

- OpenCL tutorials:  
  http://www.cmsoft.com.br/opencl-tutorial

- Kronos resources:  
  https://www.khronos.org/opencl/resources