CS4803DGC Design Game Consoles

Spring 2009
Prof. Hyesoon Kim
OpenCL

- OpenCL (open computing Language): a framework for writing programs that execute across heterogeneous platforms considering CPUs, GPUs, and other processors.
- Initiated by Apple Inc. Now AMD, Intel, NVIDIA, etc.
- AMD gave up CTM (close to Metal) and decided to support OpenCL
- Nvidia will full support openCL 1.0

Participating companies.
Processor Parallelism

CPUs
Multiple cores driving performance increase

Emerging Intersection

OpenCL Heterogeneous Computing

GPUs
Increasing general purpose data-parallel computing improving numerical precision

Multi-processor programming

Graphics APIs and Shading Languages
OpenCL standard ...

- Supports both data- and task-based parallel programming models (CPU: task, GPU: data)
- Utilizes a subset of ISO C99 with extensions for parallelism
- Defines consistent numerical requirements based on IEEE 754
- Defines a configuration profile for handheld and embedded devices
- Efficiently interoperates with OpenGL, OpenGL ES and other graphics APIs
**Impacts of openCL**

- Software developers write parallel programs that will run on many devices
- Hardware developers target openCL
- Enables OpenCL on mobile and embedded silicon
OpenCL Architecture

- Platform Model
- Memory Model
- Execution Model
- Programming Model
One Host + one or more compute devices
-Each compute device is composed of one or more compute units
-Each compute unit is further divided into one or more processing units
Execution Model

- **OpenCL Program:**
  - **Kernels**
    - Basic unit of executable code – similar to C function
    - Data-parallel or task-parallel
  - **Host Program**
    - Collection of compute kernels and internal functions
    - Analogous to a dynamic library
Execution Model

• Kernel Execution
  – The host program invokes a kernel over an index space called an **NDRange**
    • NDRange = “N-Dimensional Range”
    • NDRange can be a 1, 2, or 3-dimensional space
  – A single kernel instance at a point in the index space is called a **work-item**
    • Work-items have unique global IDs from the index space
    • **CUDA thread IDs**
  – Work-items are further grouped into **work-groups**
    • Work-groups have a unique work-group ID
    • Work-items have a unique local ID within a work-group
    • **CUDA Block IDs**
An Example of NDR

Total number of work-items = $G_x \times G_y$
Size of each work-group = $S_x \times S_y$
Contexts are used to contain and manage the state of the “world”

Kernels are executed in contexts defined and manipulated by the host
- Devices
- Kernels - OpenCL functions
- Program objects - kernel source and executable
- Memory objects

Command-queue - coordinates execution of kernels
- Kernel execution commands
- Memory commands - transfer or mapping of memory object data
- Synchronization commands - constrains the order of commands

Applications queue compute kernel execution instances
- Queued in-order
- Executed in-order or out-of-order
- Events are used to implement appropriate synchronization of execution instances
Memory Model

- Shared memory
  - Relaxed consistency
  - (similar to CUDA)
- Global memory
  - Global memory in CUDA
- Constant memory
  - Constant memory in CUDA
- Local memory (local memory to work group)
  - Shared memory in CUDA
- Private memory (private to a work item)
  - local memory in CUDA
Memory Consistency

- a relaxed consistency memory model
  - Across workitems (threads) no consistency
  - Within a work-item (thread) load/store consistency
  - Consistency of memory shared between commands are enforced through synchronization
Define N-Dimensional computation domain
- Each independent element of execution in an N-Dimensional domain is called a work-item
- N-Dimensional domain defines the total number of work-items that execute in parallel = global work size

Work-items can be grouped together — work-group
- Work-items in group can communicate with each other
- Can synchronize execution among work-items in group to coordinate memory access

Execute multiple work-groups in parallel
- Mapping of global work size to work-group can be implicit or explicit
Task Parallel Programming Model

- Data-parallel execution model must be implemented by all OpenCL compute devices
- Users express parallelism by
  - using vector data types implemented by the device,
  - enqueuing multiple tasks, and/or
  - enqueuing native kernels developed using a programming model orthogonal to OpenCL.
Synchronization

- Work-items in a single-work group
  - Similar to _synchthreads();
- Synchronization points between commands and command-queues
  - Similar to multiple kernels in CUDA but more generalized.
  - Command-queue barrier.
  - Waiting on an event.
OpenCL Framework

- **OpenCL Platform layer**: The platform layer allows the host program to discover openCL devices and their capabilities and to create contexts.
- **OpenCL Runtime**: The runtime allows the host program to manipulate contexts once they have been created.
- **OpenCL Compiler**: The OpenCL compiler creates program executables that contain OpenCL kernels.
Platform Layer

- Platform layer allows applications to query for platform specific features
- Querying platform info (i.e., OpenCL profile)
- Querying devices
  - `clGetDeviceIDs()`
    - Find out what compute devices are on the system
    - Device types include CPUs, GPUs, or Accelerators
  - `clGetDeviceInfo()`
  - Queries the capabilities of the discovered compute devices such as:
    - Number of compute cores
    - NDRange limits
    - Maximum work-group size
    - Sizes of the different memory spaces (constant, local, global)
    - Maximum memory object size
Platform Layer

• Creating contexts

    cl_context clCreateContext(cl_context_properties properties,
    cl_uint num_devices,
    const cl_device_id *devices,
    void (*pfn_notify)(const char *errinfo,
                       const void *private_info, size_t cb,
                       void *user_data),
    void *user_data,
    cl_int *errcode_ret)

  – Contexts are used by the OpenCL runtime to manage objects and execute kernels on one or more devices
  – Contexts are associated to one or more devices
  – Multiple contexts could be associated to the same device
  – `clCreateContext()` and `clCreateContextFromType()` returns a handle to the created contexts
Command-Queues

- Command-queues store a set of operations to perform
- Command-queues are associated to a context
- Multiple command-queues can be created to handle independent commands that don’t require synchronization
- Execution of the command-queue is guaranteed to be completed at sync points
Memory Objects

- **Buffer objects**
  - One-dimensional collection of objects (like C arrays)
  - Valid elements include scalar and vector types as well as user defined structures
  - Buffer objects can be accessed via pointers in the kernel

- **Image objects**
  - Two- or three-dimensional texture, frame-buffer, or images
  - Must be addressed through built-in functions

- **Sampler objects**
  - Describes how to sample an image in the kernel
  - Addressing modes
  - Filtering modes
OpenCL C for Compute Kernels

• Derived from ISO C99
  – A few restrictions: recursion, function pointers, functions in C99 standard headers ...
  – Preprocessing directives defined by C99 are supported

• Built-in Data Types
  – Scalar and vector data types, Pointers
  – Data-type conversion functions: convert_type<_sat<_roundingmode>
  – Image types: image2d_t, image3d_t and sampler_t

• Built-in Functions — Required
  – work-item functions, math.h, read and write image
  – Relational, geometric functions, synchronization functions

• Built-in Functions — Optional
  – double precision, atomics to global and local memory
  – selection of rounding mode, writes to image3d_t surface
OpenCL C language Restrictions

• Pointers to functions are not allowed
• Pointers to pointers allowed within a kernel, but not as an argument
• Bit-fields are not supported
• Variable length arrays and structures are not supported
• Recursion is not supported
• Writes to a pointer of types less than 32-bit are not supported
• Double types are not supported, but reserved
  – (Newer CUDA support this)
• 3D Image writes are not supported
• Some restrictions are addressed through extensions
## OpenCL vs. CUDA

<table>
<thead>
<tr>
<th></th>
<th>OpenCL</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Execution Model</strong></td>
<td>Work-groups/work-items</td>
<td>Block/Thread</td>
</tr>
<tr>
<td><strong>Memory model</strong></td>
<td>Global/constant/local/private</td>
<td>Global/constant/shared/local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ Texture</td>
</tr>
<tr>
<td><strong>Memory consistency</strong></td>
<td>Weak consistency</td>
<td>Weak consistency</td>
</tr>
<tr>
<td><strong>Synchronization</strong></td>
<td>Synchronization using a work-group barrier (between work-items)</td>
<td>Using synch_threads Between threads</td>
</tr>
</tbody>
</table>
CUDA ? OpenCL?

NVIDIA’s OpenCL Roadmap

C for CUDA Roadmap

http://www.geeks3d.com/?p=2582
OpenCL and C for CUDA

Entry point for developers who want low-level API

Entry point for developers who prefer high-level C

C for CUDA

OpenCL

Shared back-end compiler and optimization technology

PTX

GPU

http://www.geeks3d.com/?p=2582