Visualization of OpenCL Application Execution on CPU-GPU Systems

A. Ziabari*, R. Ubal*, D. Schaa**, D. Kaeli*

*NUCAR Group, Northeastern University
**AMD
Introduction and Motivation

- Simulators
  - Design evaluation (Pre- and Post- silicon)
  - Design validation
  - Education
  - … and much more

- Visualization
  - Complimentary to the simulator
  - Easy to interact
  - Thorough study of the simulated data

- Motivation
  - Teaching details of OpenCL application execution
Outline

• Background and simulation methodology
• OpenCL application on the host
• OpenCL application on the GPU device
• Education through visualization
• Ongoing Work
Background and Simulation Methodology
The Multi2Sim Simulation Framework

- Simulation framework for CPU, GPU, and heterogeneous systems
- Support for CPU architectures: x86, ARM, MIPS
- Support for GPU architectures: AMD Evergreen, AMD Southern Islands, NVIDIA Kepler, HSA intermediate language
- Application-level simulation

Full-system simulation

Applicaton-only simulation

a) Full-system simulator

b) Application-only simulator
• Four isolated software modules for each architecture (x86, SI, ARM, ...) 
• Each module has a command-line interface for stand-alone execution, or an API for interaction with other modules.
Outline

• Background and simulation methodology
• OpenCL application on the host
• OpenCL application on the GPU device
• Education through visualization
• Ongoing Work
OpenCL on the Host
The OpenCL CPU Host Program

- Native
  - An x86 OpenCL host program performs an OpenCL API call.

- Multi2Sim
  - Same
OpenCL on the Host

The OpenCL Runtime Library

- **Native**
  - AMD's OpenCL runtime library handles the call, and communicates with the driver through system calls `ioctl`, `read`, `write`, etc. These are referred to as ABI calls.

- **Multi2Sim**
  - Multi2Sim's OpenCL runtime library, running with guest code, transparently intercepts the call. It communicates with the Multi2Sim driver using system calls with codes not reserved in Linux.
OpenCL on the Host

The OpenCL Device Driver

- **Native**
  - The AMD Catalyst driver (kernel module) handles the ABI call and communicates with the GPU through the PCIe bus

- **Multi2Sim**
  - An OpenCL driver module (Multi2Sim code) intercepts the ABI call and communicates with the GPU emulator
OpenCL on the Host

The GPU Emulator

- **Native**
  - The command processor in the GPU handles the messages received from the driver

- **Multi2Sim**
  - The GPU emulator updates its internal state based on the message received from the driver
OpenCL on the Host

Transferring Control

• The **host program** performs API call `clEnqueueNDRangeKernel`
• The **runtime** intercepts the call, and enqueues a new task in an OpenCL command queue object. A user-level thread associated with the command queue eventually processes the command, performing a `LaunchKernel` ABI call
• The **driver** intercepts the ABI call, reads ND-Range parameters, and launches the GPU emulator
• The **GPU emulator** enters a simulation loop until the ND-Range completes
Outline

• Background and simulation methodology
• OpenCL application on the host
• OpenCL application on the GPU device
• Education through visualization
• Ongoing Work
OpenCL on the Device

Execution Model

- **Execution elements**
  - **Work-items** execute multiple instances of the same kernel code
  - **Work-groups** are sets of work-items that can synchronize and communicate efficiently
  - The **ND-Range** is composed by all work-groups, not communicating with each other and executing in any order

![Diagram showing ND-Range, Work-Group, and Work-Item with memory storage and communication levels.]
SI GPU Compute Pipelines

Compute Device

- A **command processor** receives commands and data from the CPU
- A **dispatcher** splits the ND-Range in work-groups and sends them into the compute units.
- A set of **compute units** runs work-groups.
- A **memory hierarchy** serves global memory accesses
SI GPU Compute Pipelines

Compute Unit

- The **instruction memory** of each compute unit contains a copy of the OpenCL kernel
- A **front-end** fetches instructions, partly decodes them, and sends them to the appropriate execution unit
- There is **one instance** of the following execution units: scalar unit, vector-memory unit, branch unit, LDS (local data store) unit
- There are **multiple instances** of SIMD units
Outline

• Background and simulation methodology
• OpenCL application on the host
• OpenCL application on the GPU device
• Education through visualization
• Ongoing Work
Education through Visualization

Visualization tool - Main Panel

• Cycle bar on main window for navigation
• Panel on main window shows workgroups mapped to compute units
Education through Visualization
Visualization tool - Main Panel

- Cycle bar on main window for navigation.
- Panel on main window shows workgroups mapped to compute units.
- Clicking on the **Detail** button opens a secondary window with a pipeline diagram.
• Front-end fetch and issue stages happens for every instruction
• After issue, the instruction continues in one of five different pipelines
  • Example: Vector unit has five pipeline stages; decode, read operand, memory, write to register and complete
• Each pipeline is color-coded to provide ease of differentiation
Education through Visualization

The Memory Hierarchy

- Flexible hierarchies
  - Any number of caches organized in any number of levels
  - Cache levels connected through default switch cross-bar interconnects, or complex custom interconnect configurations
- Clicking on the detail button opens a new window for the memory module
Education through Visualization

The Memory Hierarchy

- In this example:
  - 2-way set associative cache with 16 sets
- Each entry in the table is a cache block
- For each block visualization tool shows:
  - The state
  - The tag
  - Number of sharers
  - Number of in-flight accesses
Education through Visualization

The Interconnection Network

- Each message in the network is associated to an access from memory hierarchy
- Detail of the message lifetime can be followed by clicking the detail button on the main panel
- Detail button opens a window containing the network graph
- It shows:
  - The state of the links at each cycle
  - Congestions in the network due to the nature of OpenCL application
Education through Visualization

The Interconnection Network

- Information about individual nodes in the network graph can be obtained by clicking *detail* button on the node panel
  - State of the packets in the buffers
  - Occupancy of the buffers and links
Education through Visualization
The Memory Snapshot – Identifying application patterns

- Visualizing the memory access pattern of the OpenCL workload
  - Identifying temporal and spatial locality
  - Identifying scattered or non-recurring accesses
  - Identifying patterns in loads and stores
Education through Visualization

The Network Snapshot - Identifying application patterns

- Sampling network traffic
  - Identifying network bottlenecks in the OpenCL application execution
  - Finding traffic patterns in the execution of the application

![Network Snapshot Chart]

Offered Bandwidth - Sampled at 1000 Cycles

Network Bandwidth (Byte)

Time (Cycle)
Outline

• Background and simulation methodology
• OpenCL application on the host
• OpenCL application on the GPU device
• Education through visualization
• Ongoing Work
## Simulation Support

### Supported Architectures

<table>
<thead>
<tr>
<th></th>
<th>Disasm.</th>
<th>Emulation</th>
<th>Timing Simulation</th>
<th>Graphic Pipelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>x</td>
<td>In progress</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MIPS</td>
<td>x</td>
<td>In progress</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>x86</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>AMD Evergreen</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>AMD Southern Islands</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NVIDIA Fermi</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NVIDIA Kepler</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HSA Intermediate Language</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Simulation Support

Supported Benchmarks

- CPU benchmarks
  - SPEC 2000 and 2006
  - Mediabench
  - SPLASH2
  - PARSEC 2.1

- GPU benchmarks
  - AMD SDK 2.5 Evergreen
  - AMD SDK 2.5 Southern Islands
  - AMD SDK 2.5 x86 kernels
  - Rodinia
  - Parboil
Visualization Support

- HSA
  - Debugger
  - Profiler

- SimPoint
  - Fast-forwarding
  - Program phase analysis

- Accurate DRAM model

- Fault injection data

- Local memory and register file
The Multi2Sim Community

Collaboration Opportunities

• Current collaborators
  • Univ. of Mississippi, Univ. of Toronto, Univ. of Texas, Univ. Politecnica de Valencia (Spain), Boston University, AMD, NVIDIA

• Top of Trees (www.TopOfTrees.com)
  • Online framework for collaborative software development
  • Code peer reviews
  • Forum
  • Bug tracker

• Multi2Sim Project
  • 622 users registered (5/11/2015)
The Multi2Sim Community

Sponsors

![AMD](image1.png)
![Analog Devices](image2.png)
![analogic](image3.png)
![bk medical](image4.png)
![Global Enabling Sciences & Innovation](image5.png)
![NSF](image6.png)
![NVIDIA](image7.png)
![Qualcomm](image8.png)
![Samsung](image9.png)
Thank you!
Questions?