Exascale Monte Carlo R&D

Summer 2012

Larry Cox, Ryan Marcus
Overview

- **Exascale computing**
  - Different technologies
  - Getting there

- **High-performance proof-of-concept: MCMini**
  - Monte Carlo neutron transport
  - Features
  - Results

- **OpenCL toolkit: Oatmeal**
  - Purpose
  - Features
Exascale Computing

- 558 times faster than Roadrunner
- Long way off
  - Intel predicts 2018
- Lots of different technologies
- Some commonalities between all technologies
  - Host / Device paradigm
  - Parallel
Exascale Computing: Technologies

- **NVidia CUDA**
  - The original GPGPU language
  - Proprietary: tied to NVidia devices

- **Khronos OpenCL**
  - A multi-platform massively parallel standard
  - Runs on all current HPC hardware

- **Intel MIC**
  - 4 x86 CPUs packed onto a PCI card
  - Runs current code (Fortran, C) with little modification

- **Advantages and disadvantages to each**
### Exascale Computing: Technologies

<table>
<thead>
<tr>
<th>Quality</th>
<th>CUDA</th>
<th>OpenCL</th>
<th>MIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs everywhere</td>
<td>☒</td>
<td>☑</td>
<td>☒</td>
</tr>
<tr>
<td>Open standard</td>
<td>☒</td>
<td>☑</td>
<td>☒</td>
</tr>
<tr>
<td>Pre-existing code</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>Large user base</td>
<td>☑</td>
<td>☑</td>
<td>☒</td>
</tr>
<tr>
<td>CPU support</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>

#### Quality

<table>
<thead>
<tr>
<th>Backer</th>
<th>CUDA</th>
<th>OpenCL</th>
<th>MIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVidia</td>
<td></td>
<td>Apple, AMD, Intel</td>
<td>Intel</td>
</tr>
<tr>
<td>API Support</td>
<td>Good</td>
<td>Moderate</td>
<td>Full</td>
</tr>
</tbody>
</table>
Exascale Computing: Technologies

- Hardware-agnostic code is very important
  - Write once, run anywhere
  - Changing landscape of HPC means compatibility is important

- Running current code is very attractive, BUT it won’t scale to the same level
  - Memory constraints
  - Heavy instruction set

- Open standard is key: code can’t depend on one company

- MCMini’s HPC technology: OpenCL
MCMini

- A Monte Carlo “mini-app”
- Proof of concept
- Performance capable and performance driven
  - Written in C
- Basic, Newtonian physics
  - Algorithmically similar to “real” physics
- Support for three reaction types: scatter, fission, and absorption
- Reads LNK3DNT meshes (MCNP geometry compatible)
- Multiple OpenCL devices (CPU, GPU)
- Multiple nodes
MCMini: Capabilities

- MCMini can handle a lot of particles on really big geometries
- Trace and attenuation code is result-similar to MCNP
- Very capable (4th generation particles calculated)

<table>
<thead>
<tr>
<th>Nodes</th>
<th>GPUs</th>
<th>Mesh cells</th>
<th>Particles</th>
<th>Time (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1000^3</td>
<td>10^5</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2000^3</td>
<td>10^5</td>
<td>1.02</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>3000^3</td>
<td>10^5</td>
<td>2.8</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>10000^3</td>
<td>10^5</td>
<td>6.0</td>
</tr>
</tbody>
</table>
MCMini Scaling: CPU vs GPU (Cypress)
MCMMini Scaling: 1 GPU vs 2 GPUs (Cypress)
MCMi Xi Scaling: Nodes

![Graph showing the scaling of particles with time for different numbers of nodes. The x-axis represents the number of particles, and the y-axis represents time in seconds. The graph compares 1 Node, 2 Nodes, and 4 Nodes.]
MCMini Scaling: Cypress vs Tesla (C2070)

- **Particles**
- **Time (S)**

Graph showing the comparison between Cypress and Tesla for different numbers of particles.

- 1 Cypress
- 2 Cypress
- 1 Tesla
- 2 Tesla
MCMini / MCNP comparison

- Neutron flux calculations
  - “Comparable”
  - Similarly scaling

- Time comparison is inherently unfair
  - MCNP is doing far more than MCMini

- Geometries that take a long time in MCNP generally requires minutes or seconds in MCMini
MCMini: Results

- OpenCL is a viable HPC tool
  - Scaling factors
  - General performance

- OpenCL code can be easily designed to run on both CPUs and GPUs

- Monte Carlo style tasks are very well fitted to OpenCL and GPGPU computing

- Host/device paradigm introduces some interesting complications
Oatmeal: Purpose

- Traditionally, you either have enough RAM, or you don’t.

- With GPGPUs
  - Large host memory pool (500GB on some Darwin nodes)
  - Limited device memory pool (1 – 5 GB)
  - Device can’t do computations on data not in device memory

- Memory can be moved around between the host and the device
  - Costly

- OpenCL code is broken down into kernels
  - Minimal task unit: the smallest logical division of a task
Oatmeal: Purpose

- Steps require different data
  - Data required by a step is that step’s “context”

- Context switching can be expensive
  - Transferring data takes time

- We want to minimize data transfer
  - But we can’t run out of memory

- We could maintain a minimum context

- Or we can calculate an optimal context
Oatmeal: Features

- Oatmeal is the OpenCL AuTomatic Memory Allocation Library
- A C++ framework for calculating optimal context switches
- Takes in a dynamic graph, executes the program
- Can run kernels over multiple devices and automatically reduce data
- Can run host-based tasks in parallel
- Can be used to profile memory usage and identify hotspots
  - Both memory and computational
- Can determine if bandwidth costs exceed computational benefit
  - And then run code on the CPU
Conclusions

- Despite driver issues, OpenCL seems like a good, hardware agnostic tool

- MCMini demonstrates the possibility for GPGPU-based Monte Carlo methods
  - Shows great scaling for HPC application
  - Algorithmic equivalence

- Oatmeal provides a flexible framework to aid in the development of scientific OpenCL codes