Programming for Hybrid Architectures

John E. Stone
Theoretical and Computational Biophysics Group
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
http://www.ks.uiuc.edu/Research/gpu/

GPGPU 2015: Advanced Methods for Computing with CUDA,
University of Cape Town, April 2015
Solid Growth of GPU Accelerated Apps

Top HPC Applications

- **Molecular Dynamics**: AMBER, CHARMM, DESMOND, GROMACS, LAMMPS, NAMD
- **Quantum Chemistry**: Abinit, Gaussian, GAMESS, NWChem
- **Material Science**: CP2K, QMCPACK, Quantum Espresso, VASP
- **Weather & Climate**: COSMO, GEOS-5, HOMME, CAM-SE, NEMO, NIM, WRF
- **Lattice QCD**: Chroma, MILC
- **Plasma Physics**: GTC, GTS
- **Structural Mechanics**: ANSYS, Mechanical LS-DYNA, Implicit, MSC Nastran, OptiStruct, Abaqus/Standard
- **Fluid Dynamics**: ANSYS Fluent, Culises (OpenFOAM)

# of GPU-Accelerated Apps

<table>
<thead>
<tr>
<th>Year</th>
<th>2011</th>
<th>2012</th>
<th>2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>113</td>
<td>182</td>
<td>272</td>
</tr>
</tbody>
</table>

Courtesy NVIDIA
Major Approaches For Programming Hybrid Architectures

- Use drop-in libraries in place of CPU-based libraries
  - Little or no code development
  - Speedups limited by Amdahl’s Law and overheads associated with data movement between CPUs and GPU accelerators
  - Examples: MAGMA, BLAS-variants, FFT libraries, etc.

- Generate accelerator code as a variant of CPU source, e.g. using OpenMP w/ OpenACC, similar methods

- Write lower-level accelerator-specific code, e.g. using CUDA, OpenCL, other approaches
GPU Accelerated Libraries

“Drop-in” Acceleration for your Applications

Linear Algebra
FFT, BLAS, SPARSE, Matrix

Numerical & Math
RAND, Statistics

Data Struct. & AI
Sort, Scan, Zero Sum

Visual Processing
Image & Video

Courtesy NVIDIA
OpenACC: Open, Simple, Portable

- Open Standard
- Easy, Compiler-Driven Approach
- Portable on GPUs and Xeon Phi

CAM-SE Climate
6x Faster on GPU
Top Kernel: 50% of Runtime

Courtesy NVIDIA
Using the CPU to Optimize GPU Performance

• GPU performs best when the work evenly divides into the number of threads/processing units

• Optimization strategy:
  – Use the CPU to “regularize” the GPU workload
  – Use fixed size bin data structures, with “empty” slots skipped or producing zeroed out results
  – Handle exceptional or irregular work units on the CPU; GPU processes the bulk of the work concurrently
  – On average, the GPU is kept highly occupied, attaining a high fraction of peak performance
CUDA Grid/Block/Thread Decomposition

1-D, 2-D, or 3-D Computational Domain

1-D, 2-D, or 3-D (SM >= 2.x) Grid of thread blocks:

Padding arrays out to full blocks optimizes global memory performance by guaranteeing memory coalescing
Avoiding Shared Memory Bank Conflicts: 
Array of Structures (AOS) vs. 
Structure of Arrays (SOA)

• AOS:
typedef struct {
  float x;
  float y;
  float z;
} myvec;
myvec aos[1024];
aos[threadIdx.x].x = 0;
aos[threadIdx.x].y = 0;

• SOA
typedef struct {
  float x[1024];
  float y[1024];
  float z[1024];
} myvecs;
myvecs soa;
soa.x[threadIdx.x] = 0;
soa.y[threadIdx.x] = 0;
Time-Averaged Electrostatics Analysis on Energy-Efficient GPU Cluster

- **1.5 hour** job (CPUs) reduced to **3 min** (CPUs+GPU)
- Electrostatics of thousands of trajectory frames averaged
- Per-node power consumption on NCSA “AC” GPU cluster:
  - CPUs-only: 448 Watt-hours
  - CPUs+GPUs: 43 Watt-hours
- GPU Speedup: **25.5x**
- Power efficiency gain: **10.5x**

### AC Cluster GPU Performance and Power Efficiency Results

<table>
<thead>
<tr>
<th>Application</th>
<th>GPU speedup</th>
<th>Host watts</th>
<th>Host+GPU watts</th>
<th>Perf/watt gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMD</td>
<td>6</td>
<td>316</td>
<td>681</td>
<td>2.8</td>
</tr>
<tr>
<td>VMD</td>
<td>25</td>
<td>299</td>
<td>742</td>
<td>10.5</td>
</tr>
<tr>
<td>MILC</td>
<td>20</td>
<td>225</td>
<td>555</td>
<td>8.1</td>
</tr>
<tr>
<td>QMCPACK</td>
<td>61</td>
<td>314</td>
<td>853</td>
<td>22.6</td>
</tr>
</tbody>
</table>

Optimizing GPU Algorithms for Power Consumption

NVIDIA “Carma”, “Kayla”, “Jetson” single board computers

Tegra+GPU energy efficiency testbed
## Time-Averaged Electrostatics Analysis on NCSA Blue Waters

<table>
<thead>
<tr>
<th>NCSA Blue Waters Node Type</th>
<th>Seconds per trajectory frame for one compute node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray XE6 Compute Node: 32 CPU cores (2xAMD 6200 CPUs)</td>
<td>9.33</td>
</tr>
<tr>
<td>Cray XK6 GPU-accelerated Compute Node: 16 CPU cores + NVIDIA X2090 (Fermi) GPU</td>
<td>2.25</td>
</tr>
<tr>
<td>Speedup for GPU XK6 nodes vs. CPU XE6 nodes</td>
<td><strong>XK6 nodes are 4.15x faster overall</strong></td>
</tr>
<tr>
<td>Tests on XK7 nodes indicate MSM is CPU-bound with the Kepler K20X GPU. Performance is not much faster (yet) than Fermi X2090 Need to move spatial hashing, prolongation, interpolation onto the GPU…</td>
<td><strong>In progress….. XK7 nodes 4.3x faster overall</strong></td>
</tr>
</tbody>
</table>

Preliminary performance for VMD time-averaged electrostatics w/ Multilevel Summation Method on the NCSA Blue Waters Early Science System
Multilevel Summation on the GPU

Accelerate **short-range cutoff** and **lattice cutoff** parts

Performance profile for 0.5 Å map of potential for 1.5 M atoms. Hardware platform is Intel QX6700 CPU and NVIDIA GTX 280.

<table>
<thead>
<tr>
<th>Computational steps</th>
<th>CPU (s)</th>
<th>w/ GPU (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short-range cutoff</td>
<td>480.07</td>
<td>14.87</td>
<td>32.3</td>
</tr>
<tr>
<td>Long-range anterpolation</td>
<td>0.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restriction</td>
<td>0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>lattice cutoff</strong></td>
<td>49.47</td>
<td>1.36</td>
<td>36.4</td>
</tr>
<tr>
<td>prolongation</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interpolation</td>
<td>3.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>533.52</td>
<td>20.21</td>
<td>26.4</td>
</tr>
</tbody>
</table>

Multilevel summation of electrostatic potentials using graphics processing units.
Multi-GPU NUMA Architectures:

- Example of a “balanced” PCIe topology
- NUMA: Host threads should be pinned to the CPU that is “closest” to their target GPU
- GPUs on the same PCIe I/O Hub (IOH) can use CUDA peer-to-peer transfer APIs
- Intel: GPUs on different IOHs can’t use peer-to-peer

Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations
Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten.
http://dx.doi.org/10.1016/j.parco.2014.03.009
Multi-GPU NUMA Architectures:

- Example of a very “unbalanced” PCIe topology
- CPU 2 will overwhelm its QP/HT link with host-GPU DMAs
- Poor scalability as compared to a balanced PCIe topology

Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations
Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten.
http://dx.doi.org/10.1016/j.parco.2014.03.009
Multi-GPU NUMA Architectures:

- GPU-to-GPU peer DMA operations are much more performant than other approaches, particularly for moderate sized transfers.
- Likely to perform even better in future multi-GPU cards with direct GPU links, e.g. announced “NVLink”.

Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations
Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten.
http://dx.doi.org/10.1016/j.parco.2014.03.009
Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations
Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten.
http://dx.doi.org/10.1016/j.parco.2014.03.009
Single CUDA Execution “Stream”

- Host CPU thread launches a CUDA “kernel”, a memory copy, etc. on the GPU
- GPU action runs to completion
- Host synchronizes with completed GPU action

Diagram:

- CPU code running
- CPU waits for GPU, ideally doing something productive
- CPU code running
Multiple CUDA Streams: Overlapping Compute and DMA Operations

Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations
Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten.
http://dx.doi.org/10.1016/j.parco.2014.03.009
Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- NVIDIA CUDA Center of Excellence, University of Illinois at Urbana-Champaign
- NVIDIA CUDA team
- NCSA Blue Waters Team
- Funding:
  - NSF OCI 07-25070
  - NSF PRAC “The Computational Microscope”
  - NIH support: 9P41GM104601, 5R01GM098243-02
GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/


• Unlocking the Full Potential of the Cray XK7 Accelerator  Mark Klein and John E. Stone. Cray Users Group, Lugano Switzerland, May 2014.


GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/


GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/


GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/


