Adapting a Message-Driven Parallel Application to GPU-Accelerated Clusters

James Phillips
John Stone
Klaus Schulten

http://www.ks.uiuc.edu/Research/gpu/
Outline

• NAMD and message-driven programming
• Adapting NAMD to GPU-accelerated clusters
• Old NCSA QP cluster performance results
• New NCSA Lincoln cluster performance results
• Does CUDA like to share?
NAMD Hybrid Decomposition


- Spatially decompose data and communication.
- Separate but related work decomposition.
- “Compute objects” facilitate iterative, measurement-based load balancing system.
Message-Driven Programming

- No receive calls as in “message passing”
- Messages sent to object “entry points”
- Incoming messages placed in queue
  - Priorities are necessary for performance
- Execution generates new messages
- Implemented in Charm++ on top of MPI
  - Can be emulated in MPI alone
  - Charm++ provides tools and idioms
  - Parallel Programming Lab: http://charm.cs.uiuc.edu/
System Noise Example
Timeline from Charm++ tool “Projections”
NAMD Overlapping Execution

Phillips et al., SC2002.

Objects are assigned to processors and queued as data arrives.
Message-Driven CUDA?

• No, CUDA is too coarse-grained.
  – CPU needs fine-grained work to interleave and pipeline.
  – GPU needs large numbers of tasks submitted all at once.

• No, CUDA lacks priorities.
  – FIFO isn’t enough.

• Perhaps in a future interface:
  – Stream data to GPU.
  – Append blocks to a running kernel invocation.
  – Stream data out as blocks complete.
“Remote Forces”

- Forces on atoms in a local patch are “local”
- Forces on atoms in a remote patch are “remote”
- Calculate remote forces first to overlap force communication with local force calculation
- Not enough work to overlap with position communication

Work done by one processor
Overlapping GPU and CPU with Communication

One Timestep
Actual Timelines from NAMD
Generated using Charm++ tool “Projections”
NCSA “4+4” QP Cluster

2.4 GHz Opteron + Quadro FX 5600

seconds per step

CPU only
with GPU
GPU

faster

6.76 3.33
<table>
<thead>
<tr>
<th>CPU Cores &amp; GPUs</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPU-accelerated performance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local blocks/GPU</td>
<td>13186</td>
<td>5798</td>
<td>2564</td>
<td>1174</td>
<td>577</td>
</tr>
<tr>
<td>Remote blocks/GPU</td>
<td>1644</td>
<td>1617</td>
<td>1144</td>
<td>680</td>
<td>411</td>
</tr>
<tr>
<td>GPU s/step</td>
<td>0.544</td>
<td>0.274</td>
<td>0.139</td>
<td>0.071</td>
<td>0.040</td>
</tr>
<tr>
<td>Total s/step</td>
<td>0.960</td>
<td>0.483</td>
<td>0.261</td>
<td>0.154</td>
<td>0.085</td>
</tr>
<tr>
<td><strong>Unaccelerated performance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total s/step</td>
<td>6.76</td>
<td>3.33</td>
<td>1.737</td>
<td>0.980</td>
<td>0.471</td>
</tr>
<tr>
<td><strong>Speedup from GPU acceleration</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Factor</td>
<td>7.0</td>
<td>6.9</td>
<td>6.7</td>
<td>6.4</td>
<td>5.5</td>
</tr>
<tr>
<td>CPU Cores &amp; GPUs</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>60</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>GPU-accelerated performance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local blocks/GPU</td>
<td>2802</td>
<td>1131</td>
<td>492</td>
<td>216</td>
<td>98</td>
</tr>
<tr>
<td>Remote blocks/GPU</td>
<td>708</td>
<td>624</td>
<td>386</td>
<td>223</td>
<td>136</td>
</tr>
<tr>
<td>GPU s/step</td>
<td>0.051</td>
<td>0.027</td>
<td>0.015</td>
<td>0.008</td>
<td>0.005</td>
</tr>
<tr>
<td>Total s/step</td>
<td>0.087</td>
<td>0.048</td>
<td>0.027</td>
<td>0.018</td>
<td>0.013</td>
</tr>
<tr>
<td><strong>Unaccelerated performance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total s/step</td>
<td>0.561</td>
<td>0.284</td>
<td>0.146</td>
<td>0.077</td>
<td>0.044</td>
</tr>
<tr>
<td><strong>Speedup from GPU acceleration</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Factor</td>
<td>6.4</td>
<td>5.9</td>
<td>5.4</td>
<td>4.3</td>
<td>3.4</td>
</tr>
</tbody>
</table>
GPU Cluster Observations

• Tools needed to control GPU allocation
  – Simplest solution is rank % devicesPerNode
  – Doesn’t work with multiple independent jobs

• CUDA and MPI can’t share pinned memory
  – Either user copies data or disable MPI RDMA
  – Need interoperable user-mode DMA standard

• Speaking of extra copies…
  – Why not DMA GPU to GPU?
  – Even better, why not RDMA over InfiniBand?
New NCSA “8+2” Lincoln Cluster

- CPU: 2 Intel E5410 Quad-Core 2.33 GHz
- GPU: 2 NVIDIA C1060
  - Actually S1070 shared by two nodes
- How to share a GPU among 4 CPU cores?
  - Send all GPU work to one process?
  - Coordinate via messages to avoid conflict?
  - Or just hope for the best?
NCSA Lincoln Cluster Performance
(8 cores and 2 GPUs per node, very early results)

STMV s/step

2 GPUs = 24 cores
4 GPUs
8 GPUs
16 GPUs

8 GPUs = 96 CPU cores

NIH Resource for Macromolecular Modeling and Bioinformatics
http://www.ks.uiuc.edu/

Beckman Institute, UIUC
No GPU Sharing (Ideal World)
GPU Sharing (Desired)

Client 1

Remote Force

Local Force

Client 2

Remote Force

Local Force
GPU Sharing (Feared)

Client 1

Client 2

Remote Force

Local Force

Remote Force

Local Force
GPU Sharing (Observed)
GPU Sharing (Explained)

- CUDA is behaving reasonably, but
- Force calculation is actually two kernels
  - Longer kernel writes to multiple arrays
  - Shorter kernel combines output
- Possible solutions:
  - Use locks (atomics) to merge kernels (not G80)
  - Explicit inter-client coordination
Conclusions and Outlook

• CUDA today is sufficient for
  – Single-GPU acceleration (the mass market)
  – Coarse-grained multi-GPU parallelism
    • Enough work per call to spin up all multiprocessors

• Improvements in CUDA are needed for
  – Assigning GPUs to processes
  – Sharing GPUs between processes
  – Fine-grained multi-GPU parallelism
    • Fewer blocks per call than chip has multiprocessors
  – Moving data between GPUs (same or different node)

• Faster processors will need a faster network!
Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- Prof. Wen-mei Hwu, Chris Rodrigues, IMPACT Group, University of Illinois at Urbana-Champaign
- Mike Showerman, Jeremy Enos, NCSA
- David Kirk, Massimiliano Fatica, NVIDIA
- NIH support: P41-RR05969

http://www.ks.uiuc.edu/Research/gpu/