Considerations for
Scalable Heterogeneous Computing with GPUs
Philip C. Roth, Kyle L. Spafford, Jeremy S. Meredith, Jeffrey S. Vetter
Oak Ridge National Laboratory
Why Heterogeneous Systems?

• Traditional single-core CPUs were reaching thermal limits when increasing clock rates
• Multi-core allows continued performance improvements, but only so much
• Specialized devices can give higher performance, better energy efficiency
• Still need general purpose CPUs for parts of the code that cannot run on specialized devices

⇒ Heterogeneous systems – the best of both worlds
Heterogeneous Systems for Scientific Computing

• Several examples of high-profile heterogeneous systems
  – Tianhe-1A
  – Tsubame
  – Dirac
  – Lincoln
  – Roadrunner
  – Keeneland

• Many augment CPUs with GPUs
GPU Advantages

• GPU-based systems have lots of performance potential
  – High computational rate: lots of concurrency
  – High internal memory bandwidth: needed to feed all those thread contexts
  – High energy efficiency: requires less energy per flop than traditional CPUs
  – High spatial density: requires fewer nodes that CPU-based approaches to reach given performance level

• Keeneland: 201 Tflop/s peak in 7 racks
So What’s the Problem?

• Not necessarily easy to realize the potential
  – Programs must expose parallelism at many levels
  – Developers must take node and system architecture into account, not just GPU

• GPUs connected to system nodes via a relatively narrow link
  – PCIe Gen 2 theoretical peak unidirectional bandwidth (x16) is 8.0 GB/s
  – In contrast, Tesla M2070 internal memory bandwidth is 148 GB/s (ECC off)

• Need timely, efficient data transfers across PCIe
It’s a Multi-Level Problem

• Scalable, high-performance programs require attention to at least three levels:
  – Within a single process
  – Within a single node
  – Across nodes
Keeneland ID System Architecture

- Traditional cluster
- 120 compute nodes
- 4 login and service nodes
- 4x QDR Infiniband Interconnect
  - Program communication
  - Parallel file system access
- 1Gb Ethernet management network
Multi-Node Programs

- Some problems are too large to fit in memory on one node
- Some problems require more computing power than one node provides
- Systems like Keeneland support multi-node programs, e.g., using MPI
CUDA/OpenCL and MPI

• Neither CUDA nor OpenCL provide any particular support for multi-node programs...
• ...Nor do they get in the way

• CUDA/OpenCL are orthogonal to MPI
• “Easy” to take an MPI program and convert some of it to run on a GPU
CUDA and MPI

• Most people program CUDA using its Runtime API
  – CUDA kernels called using “triple chevron” syntax (e.g.,
    Reduce<<<nbblocks,nthreads>>>(idata, odata, size);
  
• Must use nvcc to compile Runtime API code because only it understands this triple chevron syntax

• Easiest to use nvcc to link CUDA code (it automatically finds right libraries)

• But it is easiest to compile and link MPI code using MPI compiler drivers (e.g., mpicc)

• So who should “win”?
CUDA and MPI (2)

• Two approaches
  – Let CUDA win
    • Determine flags needed to compile and link an MPI program (some mpirun’s support ‘showme’ flag that can help)
    • Compile/link everything with nvcc, passing necessary MPI flags
    • Difficult to use anything other than the GNU compiler
  – Let MPI win
    • Separate code that uses CUDA from code that uses MPI
    • Compile CUDA code using nvcc
    • Compile all other code using MPI drivers
    • Link with MPI driver, passing link flags for finding and using CUDA libraries (e.g., -lcudart)

• We recommend the 2nd approach
NVIDIA OpenCL

- OpenCL 1.1, installed as part of CUDA 4.1
- Supports GPUs, but not CPUs
  - NVIDIA OpenCL only supports NVIDIA GPUs
  - Filename conflicts preclude installing Intel’s OpenCL alongside NVIDIA’s OpenCL

For introduction to OpenCL
- See Khronos site (http://www.khronos.org)
- See NVIDIA (http://developer.nvidia.com/opencl)
- See slides from our tutorials (http://keeneland.gatech.edu)
OpenCL and MPI

• Unlike CUDA, OpenCL only uses calls to library functions
  – Any C/C++ compiler can compile source files containing OpenCL function calls
  – No special linker for linking OpenCL programs (but do need to include OpenCL library on link line, e.g. -lOpenCL)

• For MPI+OpenCL programs
  – Use mpicc/mpicxx or equivalent to compile and link your programs
  – NVIDIA OpenCL doesn’t install headers under /usr/include; add something like -I/sw/keeneland/cuda/3.2/linux_binary/include to compile flags
  – OpenCL libs are installed in system location, so no special -L flags needed to find libraries
It’s a Multi-Level Problem

• Scalable, high-performance programs require attention to at least three levels:
  – *Within a single process*
  – Within a single node
  – Across nodes
Single Process Performance

• Lots said elsewhere in the workshop about support for good single process performance
  – Compilers/translators
  – Libraries
    • CUBLAS, CUFFT, NVIDIA Performance Primitives
    • MAGMA
    • Jacket/libJacket
    • Thrust
It’s a Multi-Level Problem

- Scalable, high-performance programs require attention to at least three levels:
  - Within a single process
  - Within a single node
  - Across nodes
Keeneland ID Node Architecture

- Hewlett Packard ProLiant SL390s G7
Within a Node

• Avoid data transfers across PCIe whenever possible
  – Transfer data to device and leave it there
  – Don’t transfer redundant/excess data (e.g., entire matrix when boundary values are all that is needed for halo exchange)

• Ensure timely, efficient data transfers when necessary
Non-Uniform Memory Access

- Node architectures result in Non-Uniform Memory Access (NUMA)
  - Point-to-point connections between devices
  - Not fully-connected topologies
  - Host memory connected to sockets instead of across a bus

![Diagram showing CPU, RAM, I/O Hub, and GPUs with QPI and PCIe connections.]
NUMA Can Affect GPUs and Network Too

Older node architecture with single I/O hub but no NUMA effects between CPU and GPU/HCA

- DL160
- Single I/O Hub
- PCIe switch connects GPUs

Keeneland node architecture with dual I/O hub but NUMA effects

- SL390
- Dual I/O Hub
- No PCIe switch
NUMA Control Mechanisms

- Process, data placement tools:
  - Tools like libnuma and numactl
  - Some MPI implementations have NUMA controls built in (e.g., Intel MPI, OpenMPI)
How much Does NUMA Impact Performance?

- Microbenchmarks to focus on individual node components
- Macrobenchmarks to focus on individual operations and program kernels
- Full applications to gauge end-user impact
Data Transfer Bandwidth

- Measured bandwidth of data transfers between CPU socket 0 and the GPUs

![Diagram showing data transfer bandwidth between CPU and GPUs](image)
The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite

- Benchmark suite with a focus on scientific computing workloads, including common kernels like SGEMM, FFT, Stencils
- Parallelized with MPI, with support for multi-GPU and cluster scale comparisons
- Implemented in CUDA and OpenCL for a 1:1 performance comparison
- Includes stability tests


Download SHOC

• Source code

• Build and Run
    • `sh ./conf/config-keeneland.sh`
    • `make`
    • `cd tools`
    • `perl driver.pl --cuda -s 4`
  – Includes example output for Keeneland

• FAQ
SHOC Test Categories

• Performance:
  – Level 0
    • Speeds and feeds: raw FLOPS rates, bandwidths, latencies
  – Level 1
    • Algorithms: FFT, matrix multiply, stencil, sort, etc.
  – Level 2:
    • Application kernels: S3D (chemistry), molecular dynamics

• System:
  – PCIe Contention, MPI latency vs. host-device bandwidth, NUMA

• Stability:
  – FFT-based, error detection
(Level 0 Example): DeviceMemory

• Motivation
  – Determine sustainable device memory bandwidth
  – Benchmark local, global, and image memory

• Basic design
  – Test different memory access patterns, i.e. coalesced, uncoalesced
  – Measure both read and write bandwidth
  – Vary number of threads in a block

Coalesced

Thread sequential / Uncoalesced

Thread 1
Thread 2
Thread 3
Thread 4
Motivation
- Supports investigation of accelerator usage within parallel application context
- Serial and True Parallel versions

Basic design
- 9-point stencil operation applied to 2D data set
- MPI uses 2D Cartesian data distribution, with periodic halo exchanges
- Applies stencil to data in local memory

OpenCL/CUDA observations
- Runtime dominated by data movement
  - Between host and card
  - Between MPI processes
(Level 2 Example): S3D

• Motivation
  – Measure performance of important DOE application
  – S3D solves Navier-Stokes equations for a regular 3D domain, used to simulate combustion

• Basic design
  – Assign each grid point to a device thread
  – Highly parallel, as grid points are independent

• OpenCL/CUDA observations
  – CUDA outperforms OpenCL
    • Big factor: native transcendental operations (sin, cos, tan, etc.)

3D Regular Domain Decomposition – Each thread handles a grid point, blocks handle regions
SHOC Benchmark Suite

• What penalty for “incorrect” mapping?
• Rough inverse correlation to computational intensity

<table>
<thead>
<tr>
<th>Test</th>
<th>Units</th>
<th>Correct NUMA</th>
<th>Incorrect NUMA</th>
<th>% Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGEMM</td>
<td>GFLOPS</td>
<td>535.640</td>
<td>519.581</td>
<td>3%</td>
</tr>
<tr>
<td>DGEMM</td>
<td>GFLOPS</td>
<td>239.962</td>
<td>230.809</td>
<td>4%</td>
</tr>
<tr>
<td>FFT</td>
<td>GFLOPS</td>
<td>30.501</td>
<td>26.843</td>
<td>12%</td>
</tr>
<tr>
<td>FFT-DP</td>
<td>GFLOPS</td>
<td>15.181</td>
<td>13.352</td>
<td>12%</td>
</tr>
<tr>
<td>MD</td>
<td>GB/s</td>
<td>12.519</td>
<td>11.450</td>
<td>9%</td>
</tr>
<tr>
<td>MD-DP</td>
<td>GB/s</td>
<td>19.063</td>
<td>17.654</td>
<td>7%</td>
</tr>
<tr>
<td>Reduction</td>
<td>GB/s</td>
<td>5.631</td>
<td>4.942</td>
<td>12%</td>
</tr>
<tr>
<td>Scan</td>
<td>GB/s</td>
<td>0.007</td>
<td>0.005</td>
<td>31%</td>
</tr>
<tr>
<td>Sort</td>
<td>GB/s</td>
<td>1.081</td>
<td>0.983</td>
<td>9%</td>
</tr>
<tr>
<td>Stencil</td>
<td>seconds</td>
<td>8.749</td>
<td>11.895</td>
<td>36%</td>
</tr>
</tbody>
</table>

Table 3: SHOC Benchmark Results
Full Applications

• With one application task, performance penalty for using incorrect mapping (e.g., CPU socket 0 with GPU 1)
• With two application tasks, performance penalty for using mapping that uses “long” paths for both (e.g., CPU socket 0 with GPU 1 and CPU socket 1 with GPU 0)
NUMA and Network Traffic

- Have to worry about not only process/data placement for CPU and GPU, but also about CPU and Infiniband HCA
Thread Splitting

• Instead of 1 thread that controls a GPU and issues MPI calls, split into two threads and bind to appropriate CPU sockets
It’s a Multi-Level Problem

• Scalable, high-performance programs require attention to at least three levels:
  – Within a single process
  – Within a single node
  – *Across nodes*
Interconnection Network

• System nodes are connected by an interconnection network, or interconnect
• Infiniband is a common interconnect for mid- to high-end systems
• Keeneland ID uses 4x QDR Infiniband, with 40Gb/s maximum bandwidth
The Problem

• Data is in device memory of GPU on one node, needs to be transferred to device memory of GPU on another node

• Several hops:
  – Data transferred from GPU memory to GPU buffer in host memory
  – Data copied from GPU buffer to IB buffer in host memory
  – Data read by IB HCA using RDMA transfer
  – Repeat in reverse on other end

GPUDirect

- NVIDIA and Mellanox developed an approach for allowing others to access the GPU buffer in host memory
- Eliminates the data copy from GPU buffer to IB buffer
  - Eliminates two system memory data copy operations (one on each end)
  - Keeps host CPU out of the data path
  - Up to 30% performance improvement (according to NVIDIA)

GPUDirect Status

• Initially, GPUDirect packaging made deploying it unattractive
  – Required running a specific kernel version, with RPMs provided by Mellanox
  – Kernel version was quite old
  – Concerns about keeping updated to avoid security risks

• Mellanox now makes kernel patches available for some Enterprise distributions

• Mellanox working on getting GPUDirect support included as part of stock Linux kernel
GPUDirect on Keeneland

• GPUDirect not (yet) deployed on Keeneland
  – Keeneland runs CentOS 5.5
  – That distribution requires patched kernel for PAPI
  – Have not yet tested patches for PAPI with GPUDirect patches
Back to the Node: GPUDirect 2.0

- GPUDirect 1.0 improves performance for data transfers over an interconnect
- GPUDirect 2.0 improves performance for data transfers between two GPUs in the same node
- Coming in CUDA 4.0
GPUDirect 2.0

• Old way:
  – Copy data from GPU 1 to host memory
  – Copy data from host memory to GPU 2

• New way:
  – Copy data from GPU 1 to GPU2 without host CPU involvement

• Integrates well with Unified Virtual Addressing feature (single address space for CPU and 1+ GPUs)
Summary

• Scalable, high performance computing with GPU-based heterogeneous systems is possible
• Requires attention to node-level and system level
• NUMA controls and GPUDirect (both 1 and 2) are keys to good, scalable performance
For more information

• http://keeneland.gatech.edu