Many-core Acceleration for Biomedical Applications

David Kaeli
Miriam Leeser
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA

November 12, 2009
Collaborators and Participants

- Mike Murphy – NVIDIA
- John Cavazos – Univ. of Delaware
- Greg Sharp, Homer Pien, Rick Moore - MGH
- Ph.D. students
  - Rodrigo Dominguez
  - Byunghyun Jang
  - Perhaad Mistry
  - Nicholas Moore
  - Dana Schaa
- Undergrads
  - Chanelle Green, Chawandia Mack – Spelman
  - Justin White - Northeastern
GPUs are Everywhere!

- Every desktop and laptop has a GPU on board
- Graphics Processing Units – NVIDIA, AMD Firestream/Fusion, IBM Cell, Intel Larrabee
  - Cost-effective desktop supercomputing!!
- User-friendly programming interfaces and tools
  - CUDA (NVIDIA)
  - CTM/Brook+ (AMD)
  - OpenCL
A wide range of GPU applications

- 3D image analysis
- Adaptive radiation therapy
- Acoustics
- Astronomy
- Audio
- Automobile vision
- Bioinfomatics
- Biological simulation
- Broadcast
- Cellular automata
- Fluid dynamics
- Computer vision
- Cryptography
- CT reconstruction
- Data mining
- Digital cinema / projections
- Electromagnetic simulation
- Equity training
- Film
- Financial
- Languages
- GIS
- Holographics cinema
- Machine learning
- Mathematics research
- Military
- Mine planning
- Molecular dynamics
- MRI reconstruction
- Multispectral imaging
- N-body simulation
- Network processing
- Neural network
- Oceanographic research
- Optical inspection
- Particle physics
- Protein folding
- Quantum chemistry
- Ray tracing
- Radar
- Reservoir simulation
- Robotic vision / AI
- Robotic surgery
- Satellite data analysis
- Seismic imaging
- Surgery simulation
- Surveillance
- Ultrasound
- Video conferencing
- Telescope
- Video
- Visualization
- Wireless
- X-Ray
Developing a suite of Biomedical Image Reconstruction Libraries – CUDA/OpenCL

- Target applications:
  - Deformable registration - radiation oncology
  - 3-D Iterative reconstruction – cardiovascular imaging
  - Maximum likelihood estimation – Digital Breast Tomosynthesis
  - Motion compensation in PET/CT images - cardiovascular imaging
  - Hyperspectral imaging – skin cancer screening
  - Image segmentation – brain imaging
Performance of Two Imaging Applications on a GPU

- **3-D Tomosynthesis Image Reconstruction**
  - Reduces false-positive rates during breast cancer screening
  - Utilizes a limited angle tomography approach using many 2-D images to generate a 3-D image
  - Performs an iterative Maximum Likelihood Estimation for 3-D image reconstruction
  - Performance is a barrier to image-guided biopsy

- **3-D Spiral Cone-Beam Cardiac Image Reconstruction**
  - Key new approach for identifying blockage in coronary arteries
  - Performs a least squares image reconstruction
  - Involves a forward and backward projection
  - Performance is a barrier to improve image quality
Nature of breast cancer screening work:

- For each 1000 women screened with mammography:
  - ~80 (varies from 50 to 130) are called back for additional imaging - (X-ray, US, MRI)
  - ~20 are recommended for some form of biopsy
  - ~3-7 cancers will be discovered on pathology from these biopsies

- Overall this yields:
  - Reduction in US breast cancer mortality: 30%
  - Sensitivity: 85% of all breast cancers will be detected by mammography
  - Specificity: 80 false positives in 1000 screenings
  - Positive predictive value (biopsy): ~25%
Conventional 2-D Mammography

Is this good enough?
- Positive predictive value: (~25%) 3 of 4 biopsies are benign which carries a high emotional load and additional cost

Problems are caused by superimposed tissue (structure noise)
- Missed cancers (false-negatives)
- A cancer is obscured by superimposed breast tissue
- Unnecessary callbacks (false-positives)
- Superimposed normal breast tissue may look like a tumor in a 2-D mammogram
Digital Breast Tomosynthesis (DBT) — 2nd generation GE prototype

Detector:
- 300msec readout time
- 23cm × 19.2 cm area
- 100 micron pixel size

Acquisition:
- 15 projections
- 40° arc
- 15s acquisition
- Mo and Rh anodes
- same dose as CC+MLO
- 360° gantry rotation
permits all standard views
Tomosynthesis Image Reconstruction

X-ray source
(15 views)

 Detector
(1196x2304)

Set 3D volume (guess)

X-ray projections

Compute projections Forward

Correct 3D volume Backward

3D volume
(1196x2304x45)
1. Workstation
   - Single Intel quad-core Xeon 3.2 GHz
   - Multithreaded implementation
   - 4 GB of RAM

2. Cluster A – Teracluster
   - 2.0 GHz Xeon Pentium M
   - 2 CPUs per server, dual core CPUs
   - 8 GB of RAM per server
   - Gigabit ethernet switch

3. Cluster B – Opportunity Cluster
   - 3.2 GHz Xeon EMT 64 processors
   - 2 CPUs per server
   - 4 GB of RAM per server
   - Gigabit ethernet switch

4. NVIDIA 8800 GTX GPU
   - CUDA 2.0
Tomosynthesis acceleration on a GPU

Speedup and $/sec of Breast Tomosynthesis Reconstruction* on a NVIDIA GTX8800

Execution time (sec)

Number of iterations

Cost/Performance (dollars/sec)

Number of iterations

- GTX8800
- Cluster A - 8 servers
- Cluster A - 4 servers
- Cluster A - 2 servers
- Cluster B - 16 servers
- Cluster B - 8 servers
- Cluster B - 4 servers
- Workstation (Serial)

MASSACHUSETTS GENERAL HOSPITAL
What can a GPU buy you?

Reconstructions in 1.5 secs on 3 NVIDIA 280’s
Currently, coronary heart disease (CHD) is the single leading cause of death in America.

3D CT imaging can be used to identify vulnerable plaque.

Forward and backward projection of 3D helical cone beam CT have been implemented on a NVIDIA Tesla S870 multi-GPU platform using CUDA.
Impacting heart disease with GPUs

- Single GPU speedup versus multi-threaded dual-core CPU execution – 20.3x forward / 17.8x backward
- A series of optimizations applied, includes utilizing multiple GPUs – 71.3x forward / 137x backward

* Collaboration with Synho Do (MGH), Clem Karl (BU) and Homer Pien (MGH)
GPU Strengths

- Supercomputing on the desktop
- Easy to program (small learning curve)
- Many demonstrated successes accelerating complex applications
- CUDA allows us to read and write data at any location in the device memory
- Memory close to the processors (registers + shared memory)
GPU Limitations

- Porting applications to the latest-and-greatest hardware becomes a time-consuming task
  - Suggests we need to raise the abstraction level
  - OpenCL is a step in the right direction
- Many microarchitectural details are hidden
  - Performance optimization requires deep knowledge of the microarchitecture
- Better tools are needed
  - Register usage
  - Memory blocking and layout
  - Aggressive threading schemes
  - Multi-GPU exploration
- What do researchers want??
  - Semi-automatic tuning
GPU Acceleration

- Multi-GPU acceleration
- Memory coalescing and loop vectorization
- PTX optimization
- Library optimization
How can we more effectively exploit GPUs?

- We are developing a suite of biomedical imaging libraries specific to GPUs
  - Plan to target both CUDA (performance) and OpenCL (portability)
- We are expanding on our previous work on a profile-guided approach for CUDA code to guide users on the best target multi-GPU platform for the specific application

Multi-GPU Design Space Exploration

- Predict performance for GPU programs while scaling either the number of GPUs or the input data size.
- Select the optimal configuration of GPUs (distributed/multi-system or shared-memory/multi-processing, and how many) without having to purchase hardware.
- Avoid architecture-specific optimizations which limit scalability and portability to future generations of hardware.
## Requirements for Performance Prediction

### System-specific Inputs
- Network bandwidth
- PCIe bandwidth to GPU
- Disk throughput
- RAM size

### Algorithm-specific Inputs
- Communication requirements
- Reference (single-GPU) implementation

### Variables
- Number of GPUs
- Data set sizes
- GPU Configurations

---

**Model**

**Predicted execution times**
Current GPU Optimizations

- **Loop Vectorization** – targeting the vector architecture provided for on the AMD Firestream platform
- Targets linearizing data to improve the number of loops that can vectorized on AMD GPUs

```c
A[0:N] [0:M]  
B[0:N] [0:M]  
for (i1=0;ii1<N;i1++)  
For (i2=0;i2<=M;i2++)  
A[i1][i2]=B[i1][M-i2]+1;
```

Transform Array B

```c
A[0:N] [0:M]  
B[0:N] [M:0]  
for (i1=0;ii1<N;i1++)  
For (i2=0;i2<=M;i2++)  
A[i1][i2]=B[i1][i2]+1;
```

*To appear at PPOPP 2010*
Loop vectorization

- Loop Vectorization – targeting the vector architecture provided for on the AMD Firestream platform
- Targets linearizing data to improve the number of loops that can vectorized on AMD GPUs

```
A[0:N] [0:M]
B[0:N] [0:M]
for (i1=0; i1<N; i1++)
  for (i2=0; i2<=M; i2++)
    A[i1][i2] = B[i1][M-i2] + 1;
```

Transform Array B

```
A[0:N] [0:M]
B[0:N] [M:0]
for (i1=0; i1<N; i1++)
  for (i2=0; i2<=M; i2++)
    A[i1][i2] = B[i1][i2] + 1;
```

Obtained up to 11X speedup over scalar code for Livermore Loops

*To appear at PPOPP 2010*
GPU Memory Accessible in CUDA

- Mapped host memory: up to 4GB, ~5.7GB/sec bandwidth (PCIe), accessible by multiple GPUs
- Global memory: up to 4GB, high latency (~600 clock cycles), 140GB/sec bandwidth, accessible by all threads, atomic operations (slow)
- Texture memory: read-only, cached, and interpolated/filtered access to global memory
- Constant memory: 64KB, read-only, cached, fast/low-latency if data elements are accessed in unison by peer threads
- Shared memory: 16KB, low-latency, accessible among threads in the same block, fast if accessed without bank conflicts
Memory Optimizations

- Memory Selection and Coalescing on NVIDIA GPUs
- Multiple memory spaces are exposed to the program on NVIDIA GPUs – a remnant of graphics

<table>
<thead>
<tr>
<th>Memory</th>
<th>Location</th>
<th>Cached</th>
<th>Access</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>Off Chip</td>
<td>No</td>
<td>R/W</td>
<td>Thread Grid</td>
</tr>
<tr>
<td>Constant</td>
<td>Off Chip</td>
<td>Yes</td>
<td>R</td>
<td>Thread Grid</td>
</tr>
<tr>
<td>Texture</td>
<td>Off Chip</td>
<td>Yes</td>
<td>R</td>
<td>Thread Grid</td>
</tr>
<tr>
<td>Local</td>
<td>Off Chip</td>
<td>No</td>
<td>R/W</td>
<td>Thread</td>
</tr>
<tr>
<td>Shared</td>
<td>On Chip</td>
<td>N/A</td>
<td>R/W</td>
<td>Thread Block</td>
</tr>
<tr>
<td>Register</td>
<td>On Chip</td>
<td>N/A</td>
<td>R/W</td>
<td>Thread</td>
</tr>
</tbody>
</table>

- Mathematical framework developed that characterizes loop-based array iteration spaces
- Applied mapping framework to Parboil and PhysBAM programs
- Speedups ranged from 1.3X to 15X speedup
GPU Optimizations – Rematerialization in PTX

- Goal: Reduce register pressure in PTX code which should improve performance on NVIDIA GPU
- Implemented a backward list scheduler that arranges instructions within a basic block
- Performs liveness analysis and builds a data dependence graph
- The scheduler iterates through the ready list evaluating a cost function depending on the set of registers live and the use-defs of each instruction
- We rematerialize selected registers based on lifetimes and register pressure

*Presented at NVISIONS 2009*
System-wide mapping: different parts of an application may run better on the CPU or GPU

Developing general GPU solutions for multiple problem instances is difficult

Matching the architecture is important

- Select a thread hierarchy with limited shared resources (shared memory, registers)
- Exploit the characteristics of the memory hierarchy
MATLAB OpenCL API (MOCA)

- MOCA aids implementation space exploration
  - Currently binds to CUDA, but designed to work with OpenCL
- Raises the level of abstraction for faster and easier development
  - Data structures track multiple aspects of host and GPU resources
  - Functions wrap up numerous API calls into larger tasks
  - Front end catches some errors producing useful diagnostics
  - Hides different code for different memory types
  - Concentrates CUDA code for a given activity in one location
- Goal: use MOCA functionality to explore proper parameterization of GPU libraries for adaptability
Case Study: Lung Tumor Tracking

- Based on a MATLAB lung tumor tracking application by Cui, et al.
- Matches a tumor template with incoming imagery using 2D correlations – corr2() in MATLAB
- Application handles variation during respiration by using multiple tumor templates and searching a region of interest around the original template location
- Results in greater computational requirements – a 2D correlation for each template for each position in the ROI for each video frame

\[ \text{corr2}(A, B) = \frac{\sum_{M} \sum_{N} (A_{MN} - \bar{A})(B_{MN} - \bar{B})}{\sqrt{\left(\sum_{M} \sum_{N} (A_{MN} - \bar{A})^2\right)\left(\sum_{M} \sum_{N} (B_{MN} - \bar{B})^2\right)}} \]

Data Set Parameters

- Data set includes parameters that are not powers of two
  - Total computational requirements vary on corr2() calls and template sizes
- GPU implementation launches all of the 2D correlations in parallel
  - Six individual GPU kernel are used to implement the parallel corr2()
  - Current kernels can be improved: no shared memory usage and uncoalesced global memory accesses
Explored Memory Mappings

- MOCA was used to move application data into different GPU memory types
  - Frame and template data in global or texture memory
- Runtime compared to a second MATLAB implementation optimized with knowledge from studying the application
- Textured template data improved the average GPU speedup from 22 to 31 (85 to 133 max.)
  - Data locality in template accesses allows the cached texture memory to offer improved performance
Summary of MOCA

- MOCA is useful for exploring CUDA implementation space
  - Memory type selection can be an important factor
- MOCA abstractions don't hinder (but help!) performance
  - Implementation choices/optimizations are exposed to the user
  - Optimal GPU/CPU mapping is often not 1:1
    - MOCA allows implementing functionality across the CPU/GPU boundary
- Future work: data reorganization to improve GPU memory hierarchy performance, other GPU vendors, general MOCA improvements
- Focus on the optimal dimensions for parameterization and representation within a library
- Parameterized library code applicable to a range of uses and scenarios -- focus on memory as well as kernels
Ongoing Work on GPU Acceleration

- **Physics-based simulation (PHYSBAM) acceleration**
  - Surgery simulator – Simquest

- **Machine learning algorithms**
  - Medical image analysis
  - Security

- **GPU@Home**
  - Utilize idle GPUs
Summary and Future Work

- GPUs are revolutionizing biomedical computing
- Biomedical imaging applications need to be developed in portable languages and libraries
- We can quickly determine the best GPU configuration from our estimation without purchasing hardware
  - Programmer does not need to focus on low-level optimization – instead, exploit another GPU
  - Programmer can move more easily between different versions of hardware – libraries
- Future work
  - Deliver new modeling and biomedical GPU library
  - Develop libraries based on OpenCL
  - Consider a wider range of GPU/CPU configurations
For more info on GPGPUs

- Workshop on GPU Computing for Biomedical Research – 10/22 @ Harvard Medical School
- GPGPU3 - 3rd workshop focused on utilizing GPUs for general purpose computing - to be held at ASPLOS 2010
- First Workshop on Language, Compiler and Architecture Support for GPGPU - to be held at HPCA 2010 in Bangalore
- IEEE Transactions on Parallel and Distributed System special issues on Hardware Accelerators – focused on GPUs

Also check out: [http://www.gpgpu.org](http://www.gpgpu.org)
Questions?