Investigating the Use of GPU-Accelerated Nodes for SAR Image Formation

Timothy D. R. Hartley\textsuperscript{1,2}, Ahmed R. Fasih\textsuperscript{2}, Charles A. Berdanier\textsuperscript{3}, Fusun Ozguner\textsuperscript{2}, Umit V. Catalyurek\textsuperscript{1,2}

\textsuperscript{1}Department of Biomedical Informatics, \\
\textsuperscript{2}Department of Electrical and Computer Engineering, \\
The Ohio State University, Columbus, OH, USA. \\
\textsuperscript{3}Air Force Research Laboratory, \\
Wright-Patterson Air Force Base, OH 45433
Outline

• Motivation for using GPU clusters
• SAR overview
• Software for programming GPU clusters
• Backprojection implementation
• Experimental results
• Conclusions and future work
Application Motivation

• SAR image formation is time-consuming
  • Forming 2kx2k image with a small input set takes over 60 seconds on one CPU core
• SAR image formation is highly parallel
  • Each output pixel is independently computed
  • Input data can be partitioned also
• SAR datasets are often large
Hardware Motivation

Source: Nvidia

- NVIDIA GPU
- Intel CPU

- Tesla 10-series
- Tesla 8-series
- Intel Xeon Quad-core 3 GHz
- Intel Core2 Dual-core 3.0 GHz
- Intel Pentium 4 Dual-core 3.0 GHz
- Intel Pentium 4 3.2 GHz

September 23, 2006 to March 15, 2012
SAR overview

- Spotlight-mode Synthetic Aperture Radar (SAR) aims a radar beam at 'scene center'
- Records radio pulse reflections from multiple azimuth angles (1-d line projections)
1-d Line Projections

![Graph showing 1-d line projections with labels for pulse number and range (pixels).]
Image Formation

- For each input, loop over the output pixels
- For each output pixel, determine the contribution of the input line projection
• Application is decomposed into a task-graph
  • Task graph performs computation
  • Individual tasks perform single function
  • Tasks are independent, with well-defined interfaces
  • Higher-level programming abstraction

• DataCutter
  • Coarse-grained filter-stream framework
  • OSU/Maryland-bred component-based framework
  • Third-generation runtime uses MPI for high-bandwidth network support
• Imaging pipeline composed of three coarse-grained filters connected by data streams
• 'Form Partial Image' filter is the time-consuming task = perform on GPU
• To map to a GPU cluster for even faster processing, we need to partition work

• Partition Input (PI)
  • Simple to partition
  • Input dataset consists of vectors of range profiles

• Partition Output (PO)
  • Simple to partition
  • Output dataset consists of image pixels
Partitioning Input

- Partition input into equal pieces based on number of 'Form Partial Image' filters
- Send input partitions to downstream filters
- Image formation filters output whole range of image pixels with partial results
- Aggregate final image by accumulation partial results
Partitioning Output

- Partition output from 'Form Partial Image' filters
- Broadcast input from 'Read Input Data' filter
- Each image formation filter only outputs portion of whole output image
- Aggregate final image by simple memcpy
Combining DataCutter and CUDA

• DataCutter uses a simple API
  • `init()`, `process()`, `finish()` functions
  • `process()` function usually implemented as loop
    • Read in data from upstream
    • Process data somehow
    • Write data to output stream

• CPU implementation inline in `process()` function

• CUDA implementation a function call
  • `gpu_backproj()` (for example)
  • DataCutter provides access to DCBuffer memory area with pointers – pass to CUDA function
1 process() {
2     // ... setup constants, read global values from runtime ...
3     DCBuffer * buffer;
4     while((buffer = read("in") != NULL) { 
5         // ... get data from buffer about data size ...
6
7         // ... get ptr and increment extract index ...
8         phd.real = (float *) buffer->getPtrExtract();
9         buffer->incrementExtractPointer( ... );
10
11         // ... prealloc. outgoing buffer and get ptrs ...
12
13         gpu_backproj( ... );
14     } 
15 }
CUDA Backprojection

- Fairly straightforward triple-loop computation
  - Threads calculate one pixel's values based on all input projections
  - Thread blocks are rectangular sub-images
- Interesting wrinkles
  - Line projections and sensor location information can be stored as textures
    - Leverage texture cache, which is faster than global memory
    - Leverage linear interpolation
      - Required because seldom will pixel centers fall directly on a line projection sample
  - 32 KB shared memory used to store sub-images
Experiments: System

• Perform tests on Ohio Supercomputer Center's BALE cluster

• BALE nodes
  • 2x AMD dual-core Athlon CPUs
  • 2x NVIDIA Quadro 5600 GPUs
    • 1.5 GB memory
    • G80-based (CUDA compute capability 1.0)
  • 4 GB main memory
  • Infiniband NICs
Experiments: Input and Output

- GOTCHA input dataset
  - Air Force Research Lab's Sensor Data Management System
  - SAR phase history data collected with a 640 MHz bandwidth
  - Multiple elevation angles (we only make use of one in our experiments)
  - Eleven azimuth angles
  - Parking lot with various cars and construction vehicles

- Three output image sizes (square)
  - 512 – SM, 2048 – MED, 4096 - LRG
GOTCHA Images
Experiments: Implementations

- C/MPI implementation
  - Very simple multi-process version
  - No SIMD, other optimizations
- DataCutter/C++ implementation
  - Use kernel from C/MPI version
  - Multithreaded, distributed
- C/CUDA implementation
  - Single GPU
- DataCutter/CUDA implementation
  - Multithreaded, distributed, multi-GPU
CPU Scalability Results

- Experiments run with one degree of input
- DataCutter scales slightly better than MPI
  - Due to better overlap between computation and communication
Single GPU Results

- One degree of input
- DataCutter introduces small overhead
  - Due to process invocation, higher-level paradigm, etc.
- GPU execution times scale more than 2x better than linearly with number of pixels
CPU/GPU Scalability

One degree of input, 4Kx4K (LRG) image size

Begin to see divergence on GPUs for input and output partitioning
• 11 degrees of data (largest dataset)
• Good scalability up to 8 GPUs
• Much better scalability with output partition
Conclusions and Future Work

• DataCutter is appropriate for coarse-grained GPU cluster applications
  • MPI-based runtime uses high-speed interconnects; ready for HPC applications
  • Encapsulated GPU filter code means easy application development, usage of heterogeneous systems

• Future work
  • Fix bottlenecks for increased scalability
    • Tree-style reduction
  • GT200-based GPUs -> zero-copy and simultaneous communication and computation
  • Automatic data buffer sizing
• Research at the HPC lab is funded by

• Questions?

Thanks