GPU Accelerated Web Map Services

Matthew Grossman
Mentor: Rick Brownrigg
Outline

• What is a WMS?
• JOCL/OpenCL
• Wavelets
• Parallelization
• Implementation
• Results
• Conclusions
What is a WMS?

• A mature and open standard to serve georeferenced imagery via HTTP request
• Used by virtually all map-based web sites
• Must manage GBs/ TBs of data in a variety of scales, extents, projections, and formats
• Likely to need to compose request from multiple sources
• Must return request in reasonable web response times
NASA’s WMS

- Open source, java based
- Basis of NASA’s WorldWind Production Server

- Provides a convenient framework within which to incorporate and test acceleration strategies
- Handles parsing, interpreting requests, delivering responses, configuration, etc.
Open CL and JOCL

- Abstracts GPUs and multicore CPUs to allow the user to run parallel processes
- Portable and non-vendor specific
- JOCL: open source wrapper around OpenCL
- Allows calls to OpenCL functions from Java code
- Convenient since NASA WorldWind is in Java
Wavelets

- Replaces the image pyramid which involves storing multiple copies at different resolutions and serving at the resolution closest to the request
- We used Haar basis wavelets
- Only one copy of data required
- Provide multi-resolution capabilities, without incurring additional storage overhead
- Allow you to read no more than necessary to fulfill requested resolution
- Image reconstruction can be performed in parallel per pixel
- More computation is required because of the reconstruct step.
2D Wavelet Example
Test Data

- TrueMarble dataset
- 32 files, each spanning 45x45 degrees, at 21600x21600 resolution
- Original dataset is 45GB+
- Preprocessed into wavelet encoding → 64GB
- Slightly larger because an alpha channel was added, alpha values denote the level of transparency
Opportunities for Parallelism

- Per request (WMS already dispatches a thread per request; WorldWind typically dispatches requests 4 at a time).
- Wavelet decoding (per-pixel parallelism)
- Compositing/rescaling imagery from component pieces (per-pixel parallelism)
Implementation

- Tested three implementations (though there were many more intermediate implementations)
- First implementation read TIFF files and used GDAL to process imagery to requested extent and resolution (Equivalent to NASA’s WorldWind production server)
- Second reconstructed wavelet data on the OpenCL device but resized and assembled imagery on the CPU in Java (Pipelined Generator)
- Third reconstructed, resized, and assembled the response on the OpenCL Device, avoiding the memory overhead of going back and forth between main and GPU memory (Fully Pipelined Generator)
Test Platforms

- Two platforms, four CL Devices
- Core 2 Duo @ 3.06 GHz with HDD
- Core i7 @ 2.6 GHz with SSD
- NVIDIA GeForce 9600M, 32 Cores @ 120 MHz (very slow), with HDD
- NVIDIA GeForce 950M, 384 Cores @ 900 MHz (still pretty slow by modern standards), with SSD
Results

- Collected a large amount of data using a test client to request images of varying size (resolution in pixels) and extent (portion of the whole earth image)
- Time to look at graphs!
• Request Resolution 512 x 512
• Core i7 was by far the fastest test platform
• GeForce 9600M was by far the slowest
• Other two in middle
• Request resolution 512 x 512
• For requests with an extent greater than about 8 degrees the pipelined generators were faster than GDAL
• We see the Core i7 is MUCH faster, particularly with the pipelined generators
• In large part due to using a SSD instead of HDD
• Curiously, the pipelined generator is slightly faster than the fully pipelined generator
• Request Resolution 512 x 512
• When running the generator on the actual GPU, the fully pipelined generator is faster, as expected
• The fully pipelined generator adds a few extra steps to avoid passing data back and forth between main and GPU memory
• This makes it faster when actually running on the GPU
• Since the CPU still uses main memory when run as OpenCL device it is slower to use fully pipelined generator
- Request Extent 16x16 degrees
- Diminishing advantage as request resolution increases
- More pixels, thus more threads launched
- Returned response is closer to native source image resolution, helping GDAL, while pipelined generators must do same amount of work
• Solid state drive makes a HUGE difference
• Reads are about the same, seeks can be 100+ times faster
• Requesting a large area at low res, many disk seeks, small reads. Inefficient
• Extent increases, larger area, more seeks, smaller reads
• Resolution increases, same number of seeks, longer reads, more computation. Not as bad a bottleneck
Conclusions / Further Work

• In most cases the developed implementations are faster than the original GDAL based methods
• The advantage is greatest for requests with a large extent and low resolution
• However, many requests are for small extent, high resolution data. GPU implementation performs similarly or slightly slower in these cases
• Should be tested on a high-end GPU as would be used in a dedicated server
• Advantage likely to be far greater on a GPU such as the NVIDIA K20 (2496 cores @ 706 MHz).
Thank You!

• Questions?