Using a GPU in InSAR processing to improve performance

Rob Mellors, ALOS PI 152
San Diego State University
David Sandwell
University of California, San Diego

ALOS symposium, Kona, Hawaii Nov. 2009
What is a GPU? (Graphic Processor Unit)

A graphics card that can also be used as a co-processor for numerical calculations in addition to CPU(s).

- capable of running multiple simultaneous threads (100’s to 1000’)
- well-suited for parallel processing
- can drive display as well
- cost: from $35 to $2000
- easy to install on workstation
(may be more difficult on server)
Other people use them successfully and claim great improvements

Use of Graphical Processing Units has yielded large improvements in performance for specific, parallelizable problems.

<table>
<thead>
<tr>
<th>Application</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflection seismic AVO</td>
<td>66X</td>
</tr>
<tr>
<td>Molecular dynamics</td>
<td>21X</td>
</tr>
<tr>
<td>MRI processing</td>
<td>245X</td>
</tr>
<tr>
<td>Cloud simulation</td>
<td>50X</td>
</tr>
</tbody>
</table>

http://www.ddj.com/cpp/207200659

SAR and InSAR processing is usually parallelizable
Can we do the same? (yes, but with some work)

ALOS symposium, Kona, Hawaii Nov. 2009
Question: How effective are low-cost graphical processor units (GPU’s) in improving InSAR processing performance?

- what is potential performance increase?
- is there an impact on precision?
- what is the programming effort?
- potential
- focus on low-cost (< $300 U.S.)

Overview of GPU types
Issues
Programming
Initial testing on typical InSAR computational tasks
Potential

ALOS symposium, Kona, Hawaii Nov. 2009
Motivation

ALOS baseline control and ALOS coherence means large number of possible pairs (especially at a ‘supersite’) => Requires fast processing

Ultimate goal: 1 igram per minute on a Linux workstation.....
Some related work

Use of GPU’s is expanding rapidly
Plethora of papers on various computational aspects

Several on SAR and InSAR – these are examples

SAR processing

*Clemente et al* (2009)
“SARACUDA” range-doppler 15X
specialized fft

phase unwrapping

*Karasev et al.* 5 – 35X faster
*Mistry et al.* (2009) 7.3X faster
Goals

- Test use of a low-cost GPU
- Evaluate performance versus programming effort
Primary testing machine

**Computer:**
- Intel Corei7 2.7 GHz
- Linux (Fedora 11)
- 12 GB memory
- gcc compiler

**Video card (GPU):**
- GPU: GeForce GTX 260 (~U.S. $200)
- CUDA version 2.3
- 896 MB memory
- 27 multiprocessors
- 216 cores
- 512 threads per block

Also used as driver for monitor

ALOS symposium, Kona, Hawaii Nov. 2009
Several options available for GPU programming

Compute Unified Device Architecture (CUDA) – Nvidia’s free libraries, code, compiler and documentation (support C and Fortran) FREE

OpenCL – effort designed to provide generic libraries and code for all types of GPU’s (e.g. AMD, Intel) FREE

GPU support on Matlab (GPUmat) FREE

Other commercial (rapidmind, Portland) NOT FREE

This work uses CUDA

http://www.nvidia.com/object/cuda_home.html#
CUDA Programming

Multiple levels of use

- Easy: library call (cufft, cublas)
- Harder: create kernel (with C-code like syntax)
- Optimal: kernel with shared memory
- Large set of demo code and tutorial provided
- C or Fortran
- Three downloads (drivers, compiler, and demo) available from NVIDIA CUDA site.
- Linux, Windows, and Mac

Matlab

- Create MEX files using GPU code (usually faster)
- GPUMat – allows Matlab calls to access card

from NVIDIA CUDA Compute Unified Device Architecture Programming Guide 1.1

ALOS symposium, Kona, Hawaii Nov. 2009
Issues

Must copy data from CPU to GPU
Must copy results from GPU to CPU
  • This transfer takes time
  • Best performance for intensive computation relative to data
  • Some computations might be slower on GPU
  • 1D FFT about the same as CPU
  • 2D FFT is usually faster than CPU

Handle only floating point; may have reduced precision
If GPU card is also used for display, 5 second limit on computations on the card (not CPU)
[possible to use two cards; or purchase Tesla unit designed for numerical computation]
Interferometric processing code developed primarily by UCSD/SIO/SDSU group
99% C-based and uses GMT (Generic Mapping Tools) for image manipulation
Recent effort has been on ALOS processing
Can handle ScanSAR

Range-doppler SAR
Alignment with cross-correlation
Filtering & phase difference
Geocoding
{use alternate unwrapping code}
CUDA: Library calls

Top-level libraries

-\textbf{cuFFT}:
  - mimics most FFT library calls
  - simple to include
  - use standard compilation

-\textbf{cuBLAS}:
  - mimics BLAS
  - simple

Disadvantages:
  - not optimized for specific problem
  - data transfer significant for 1D problems
cuFFT

Similar to other FFT libraries
1) allocate memory
2) create FFT plan
3) execute plan
   - real and complex, 1D, 2D, 3D
   - supports batches of 1D FFT

```c
/* create plan */
cufftPlan2d(&plan, N, N, CUFFT_C2C);

/* allocate on device */
cudaMalloc((void**)&a_d, nBytes);

/* copy data to device */
cudaMemcpy(a_d, (cufftComplex*)in, nBytes, cudaMemcpyHostToDevice);

/* execute FFT */
cufftExecC2C(plan, a_d, a_d, CUFFT_FORWARD);

/* copy results from device to host */
cudaMemcpy((cufftComplex*)in, a_d, nBytes, cudaMemcpyDeviceToHost);
```
FFT approximate benchmarks

run forward and inverse fft 1000 times

<table>
<thead>
<tr>
<th>N</th>
<th>1D fftwf</th>
<th>1D cuda</th>
<th>N</th>
<th>2D fftwf</th>
<th>2D cuda</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0</td>
<td>0.38</td>
<td>16</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>256</td>
<td>0.02</td>
<td>0.4</td>
<td>256</td>
<td>2.11</td>
<td>1.21</td>
</tr>
<tr>
<td>512</td>
<td>0.03</td>
<td>0.41</td>
<td>512</td>
<td>9.52</td>
<td>4.32</td>
</tr>
<tr>
<td>1024</td>
<td>0.08</td>
<td>0.42</td>
<td>1024</td>
<td>41.42</td>
<td>15.57</td>
</tr>
<tr>
<td>2048</td>
<td>0.17</td>
<td>0.44</td>
<td>2048</td>
<td>181.57</td>
<td>62.17</td>
</tr>
<tr>
<td>4096</td>
<td>0.37</td>
<td>0.49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>0.81</td>
<td>0.57</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>1.72</td>
<td>0.73</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1D needs roughly 8192 elements to improve performance
~2X for 16384 elements

2D performance better

error (inverse fft – original) for CUDA was larger than fftwf and about twice that of fftpack.

ALOS symposium, Kona, Hawaii Nov. 2009
Image alignment

- align master and slave images
- Two ALOS FBS single-look complex images
- calculate cross-correlation in frequency domain
- programming change minimal
  - replace fft subroutine call with cufft
  - add cufft libraries on compilation
  - should add device detection call to avoid failure on machines with no GPU (or disabled)
- execution time improved by 1.3 X
- results identical indicating that precision was not an issue
- use of cuda fft libraries will improve performance but significant (> 5X) performance increases with arrays sizes common in InSAR are unlikely.
- improved GPU fft libraries may be forthcoming

ALOS symposium, Kona, Hawaii Nov. 2009
range interpolation

Interpolate FBD signal to FBS to allow mixed interferograms
Interpolate from 6152 range bins to 11304 range bins
Use cuFFT lengths with $N = 8192$ and $16384$ (batchsize = 1)
CUDA FFT was 2X times faster
Results essentially identical
Increasing batchSize > 1 would help, but would require more extensive program modification.

ALOS symposium, Kona, Hawaii Nov. 2009
cuBLAS

- contains most of the calls of the BLAS package
- code structure similar
- multiples of 32 work better

increase in performance by 16X for 8192x8192 matrices

Big improvement!

<table>
<thead>
<tr>
<th>N</th>
<th>BLAS</th>
<th>cuBLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>0.244</td>
<td>0.245</td>
</tr>
<tr>
<td>1024</td>
<td>0.37</td>
<td>0.245</td>
</tr>
<tr>
<td>2048</td>
<td>1.199</td>
<td>0.312</td>
</tr>
<tr>
<td>4096</td>
<td>7.92</td>
<td>0.727</td>
</tr>
<tr>
<td>8192</td>
<td>62.31</td>
<td>3.74</td>
</tr>
</tbody>
</table>

ALOS symposium, Kona, Hawaii Nov. 2009
CUDA and Matlab

1) Can create “mex” files using C code that can be called from Matlab
   - Use CuBLAS or CuFFT instead of Matlab functions
   - Write kernel to execute commands
   - Need to allocate shared memory for full performance
   - up to 40X times faster than non-CUDA mex file (and 400X script)

2) GPUmat – Free package that can be installed on Matlab
   - Specific calls within Matlab
   - Generalized so uses global memory
   - Was not particularly faster

```
A = GPUsingle(rand(100,100));  % A is on GPU
C = A*A;  % multiply on GPU
Ch = single(C);  % transfer from GPU to CPU
```
Using a CUDA kernel

write code that runs on the GPU itself
1) create function (kernel)
2) call from main using \texttt{<<< >>>()} syntax
3) compile with \texttt{nvcc} (provided) compiler

```c
__global__ void addVector(float *A, float *B, float *C) {
    int i = threadIdx.x + blockDim.x * blockIdx.x; /* provided by CUDA */
    C[i] = A[i] + B[i];
}

int main(int argc, char **argv) {
    dim3 grid( CUDA_BLOCKS, 1, 1);
    dim3 threads( THREAD_NUM, 1, 1);

    cudaMemcpy(d_A, h_A, sizeof(float)*N, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, sizeof(float)*N, cudaMemcpyHostToDevice);

    addVector <<< grid, threads >>> (d_A, d_B, d_C);
    cudaMemcpy(h_C, d_C, sizeof(float)*N, cudaMemcpyDeviceToHost);
}
```

adapted from http://www.ncsa.illinois.edu/UserInfo/Training/Workshops/CUDA/presentations/tutorial-CUDA.html

ALOS symposium, Kona, Hawaii Nov. 2009
Geocoding:
kernel to calculate distance between x,y,z coordinates

Need to use shared memory – in progress

Each thread runs the kernel
All threads run simultaneously
Threads are grouped into blocks
Blocks are grouped into grid

global memory (100’s slower than shared memory)
[also texture memory]

dim3 threadIdx

dim3 blockIdx

dim3 blockDim

built in variables for thread identification
How to optimize shared memory

Visual profiler show time between computation and memory transfer

Need to allocate in kernel in code

courtesy F. McDonald, SDSU

ALOS symposium, Kona, Hawaii Nov. 2009
Test case: NLS

Test case: Finite difference non-linear Schrödinger equation CUDA kernel using shared memory allocation
Compiled as MEX files and accessed via Matlab on Windows machine

Increase over non-CUDA mex file (mex file 10X faster than script)

<table>
<thead>
<tr>
<th></th>
<th>Approximate cost</th>
<th>Regular MEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>8400GS</td>
<td>$30</td>
<td>4X</td>
</tr>
<tr>
<td>GTX275</td>
<td>$220</td>
<td>100X</td>
</tr>
</tbody>
</table>

*Courtesy R. Caplan, SDSU*

Note: if you buy a card, make sure it is CUDA capable (check Nvidia website)
Future Plans

1) Upgrade SAR processor to use batch mode cuFFT
2) Implement kernel-based code using shared memory
   - geocoding routine in progress
   - And then?
   - these kernels would likely be adaptable to most InSAR codes
   - (development would be faster with funding....)

ALOS symposium, Kona, Hawaii Nov. 2009
Conclusions

- Use of GPU has significant potential to improve performance of InSAR packages
- Use of high level libraries (such as cuFFT) are straightforward to implement and provide some improvement
- For Matlab, use Mex files
- Loss of precision does not appear to be an issue
- More dramatic improvements require specific kernels with more programming effort
- Dramatic performance increases of at least 5X and perhaps more should be possible for the entire processing stream.
- Plus the graphics card is good to display images anyway.

The use of multiple core/threaded architecture is increasing and efforts made now will likely pay off in future.

ALOS symposium, Kona, Hawaii Nov. 2009
Thank you to JAXA-EORC and ASF

(and the SDSU Computational Science 670 class)