Using Graphics Processors to Accelerate Synthetic Aperture Sonar Imaging via Backpropagation

Daniel P. Campbell, Daniel A. Cook
Georgia Tech Research Institute / Sensors and Electromagnetic Applications Laboratory
{dan.campbell, dan.cook}@gtri.gatech.edu

***ABSTRACT*-This paper describes the use of graphics processors to accelerate the backpropagation method of forming images in Synthetic Aperture Sonar (SAS) systems. SAS systems coherently process multiple pulses to provide a higher level of detail in the resolved image than is otherwise possible with a single pulse . Several models are available to resolve an image from the pulse return data; the backpropagation model is the most accurate and flexible, but also the most computationally intensive. Less flexible and accurate algorithms are frequently used because the time to resolve an image via backpropagation is held to be intolerable. A GPU-based implementation of backpropagation was developed at GTRI and inserted into sonar and radar algorithm research testbed systems. The GPU accelerated implementation formed a 4000 x 4400 SAS image from 60 seconds of sonar data in 7 seconds using 8 GPUs. This was 275x faster than a C-based implementation executing on an 8-core i7 platform, and provides an otherwise time-prohibitive technique to sonar and algorithm researchers for use in prototyping environments**

I. SONAR BACKPROPAGATION

The classical approach to both radar and sonar synthetic aperture image reconstruction is known as backprojection, or backpropagation. The scheme is straightforward. Each point in the scene being imaged contributes reflections to throughout the recorded data. Any point has a corresponding locus of echo returns in the observed data. In order to compute the value of a single output pixel in the reconstructed image, all that is required is to integrate the data along this locus while multiplying by the complex conjugate of the expected locus[2]. For each output point this operation has the form of an inner product, and the reconstructed image can the thought of as resulting from a spatially-varying correlation operation. The inner product serves as a measure of how similar the measured data is to the expected locus, and a strong correspondence results in a bright output pixel. To obtain the equation for the backpropagation, consider the ideal locus for a single point scatterer at $\mathbf{x} = [x\ y\ z]^T$, denoted by $\ell(t, u, \mathbf{x})$, which also depends on the time of recording t and the along-track position of the sensor u. The integration is performed over the locus given by the curve L:

$$f(\mathbf{x}) = \int_L \overline{E}_\delta(\ell) E_c(\ell)\, d\ell.$$

As a concrete example, consider the ideal case in which the sensor trajectory is a straight line. The resulting locus L is a hyperbola:

$$L = \sqrt{x^2 + (y-u)^2}\ .$$

This straight-line style of collection is known as stripmap imaging. It is a special case for which there exists an extremely fast Fourier-based reconstruction technique called the ω-k algorithm[3]. The name comes from the fact that it operates in the two-dimensional frequency transform domain of the data. The primary limitation of ω-k is the fact that a separate motion compensation step must be performed if the sensor does not fly a perfectly straight line.

Motion compensation generally is not a problem for small perturbations of the flight path, but large distortions result in useless data. This is illustrated in Fig. 1 which shows a typical SAS mission. The robotic vehicle is assigned a region of sea floor to survey, which it does by passing back and both over the area using the sonar to collect data from both port and starboard sides. At the end of a straight leg, the vehicle turns around to line up for the next leg. Using imaging software based on ω-k means that the data in the maneuvers cannot be used at all. Although the drawing in Fig. 1 is not to scale, the maneuvering area can represent a substantial lost opportunity for image collection because battery life limits the duration of the mission for the vehicle.

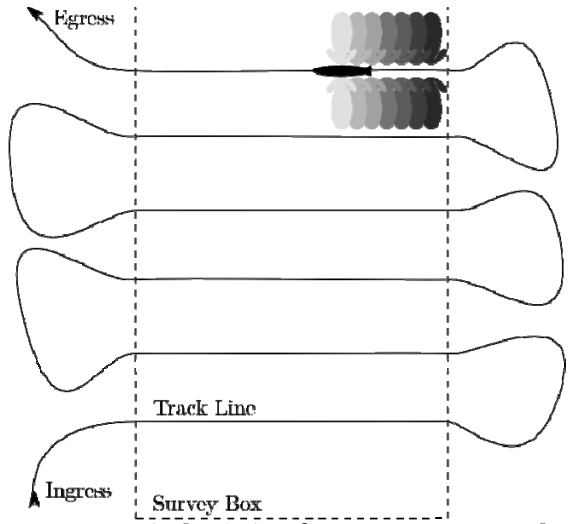

Figure 1: Typical mission for autonomous synthetic aperture sonar data collection. The ends of the 'dog bone' tracks represent lost opportunities for image generation when traditional Fourier-based techniques are used.

A second weakness of the motion compensation required for ω-k imaging is the fact that the compensation works best for sonars with narrow beams. Wide beam sensors are also of interest, but the realities of motion compensation processing can limit their operating envelope in terms of tolerable deviations from straight-line flight[1]. Lastly, the typical platform speed (1.5 m/sec). is an appreciable fraction of speed of sound underwater (1,500 m/s). Consequently, image reconstruction algorithms should account for the fact that the sensor can move between the times of transmission and reception.

Using backpropagation for image reconstruction bypasses the limitations described above and enables the achievement of the best quality imagery that can be produced by the sensor. While computationally expensive,

the use of backpropagation actually simplifies the processing chain because it obviates the need for several pre- and post-processing steps. Most notably, backpropagation naturally incorporates both motion compensation and georeferencing of the image to any desired coordinate system.

II. Accelerated Application

GTRI pursues radar and sonar signal processing algorithm research and maintains a MATLAB-based synthetic aperture processing toolbox that supports the study of new approaches for knowledge extraction. This toolbox includes a wide variety of well-understood primitives to support experimental and prototype radar and sonar processing approaches, including backpropagation. For any but the smallest output images, execution time for the backpropagation model are prohibitively long. For example, an exemplar 4400 x 4000 image formation was estimated to take over 50 days with the toolbox executing on an Intel i7 based platform. Such lengthy runtimes limit the ability of algorithm researchers to study approaches that include this approach. Algorithm prototyping must be done with much smaller images, inferior image formation approaches, or with long test and iterate cycles.

In order to improve the ability of algorithm researchers to explore new approaches for sonar and radar processing, and to facilitate transition to deployable systems, we created an accelerated implementation of the backpropagation image formation module of the synthetic aperture toolbox. The accelerated implementation exploits nVidia GPUs to form large images much more rapidly than the MATLAB implementation, allowing algorithm researchers to maintain a rapid prototype cycle while working with large images, using the most flexible and physically accurate image formation approaches.

III. GPU Background

Graphics Processing Units (GPUs) are specialized accelerators designed for rendering three dimensional graphics. The highly parallel nature of graphics rendering has allowed GPU vendors to improve capabilities over product generations by increasing on-chip parallelization more effectively than have general purpose processors. Recent changes in the programming model for GPUs have made their computing capabilities available to non graphics tasks. The high volume and competitive nature of the video game hardware and software market has driven GPU performance improvements, while holding costs low, making GPUs a very attractive accelerator for many applications.

IV. Accelerated Implementation

Synthetic aperture sonar image formation via backpropagation is well suited to parallelization on many platforms. While the mathematical operations to form the image are embarrassingly parallel, each input datum is used by many output points, and the relationship between output point location and input data used is non-linear. Because of these factors, it is straightforward to create an implementation of backpropagation that benefits from increased parallelism, but challenging to fully optimize it for a specific platform.

Despite the non-linear mapping of input data to output points, the relationship is spatially coherent. For a given pulse of return data, nearby output pixels will have similar total propagation distance, and the therefore depend on sample points that are the same as, or adjacent in memory to, each other. This relationship enables backpropagation implementations to benefit from traditional automatic caching, and to particularly benefit from caching structures that are organized for two dimensional coherency, such as those used in GPUs.

Our implementation of the SAS image formation module is accelerated with nVidia GPUs, and was developed using CUDA. The implementation launches one thread per output pixel location, with two-dimensional blocks to improve intra-block range (and thus cache) coherency. Each thread loops over the input pulses, synchronously within a block, and accumulates the correlated output as it traverses the input set. Linear interpolation of return data is implemented with texture sampling operations in order to exploit dedicated hardware on the GPUs. Our implementation accelerated the formation of a sample image with 4400 x 4000 output pixels, based on 4000 input pulses from an estimated 50 days using the original toolbox implementation to 40 seconds on an nVidia 280GTX-enhanced workstation, and under 7 seconds using a dual Tesla S1070-enhanced server platform.

The initial, naïve implementation yielded a significant speedup from the MATLAB version, but we added several additional optimizations to further improve execution times. Some of the most significant are described below.

The most significant improvement came from the elimination of register spills. The current CUDA toolchain uses global memory, which has access latency of several hundred cycles, as register spill space, instead of attempting to use shared memory. Because the number of registers used by a kernel can limit the number of concurrent threads on a multiprocessor, it is often beneficial to instruct the CUDA compiler to limit register count to a specified number. In our case, we found that this practice led to register spills, causing inner loop variables to be temporarily stored in global memory, slowing execution considerably. Instead, to reduce register usage, we selectively changed local variables into shared memory arrays. Additionally, we found that certain (but not all) calls to trigonometric functions resulted in register spills. These were corrected by using the reduced precision equivalents, such as __cosf (x), in place of cos(x). We found that the reduced precision functions did not visibly degrade the quality of the output image, but did not undertake analysis of the resulting changes in output.

We used texture sampling to improve the performance of the linear interpolation. Once the total propagation distance for a pixel-pulse has been calculated, that range must be checked against the upper and lower bounds of the pulse return data, and then converted to a real-numbered sample index. The return samples are linearly interpolated to provide an estimated return for the pixel under consideration. Done explicitly, this step requires 6 flops and 2 memory reads for each of real and imaginary return data. This operation is nearly identical to graphical texture mapping, and can leverage dedicated logic on GPUs to accelerate execution. Furthermore, texture samples are cached in CUDA, which is well suited to this operation since threads within a block have similar range to and from the sonar platform, and thus require the same or nearby input data.

GPU memory accesses can be very expensive relative to math operations. Because of the very high degree of parallelism, the cost of memory access in foregone flops can be very high if sufficient latency hiding is not available. As a result, the balancing point between profitable use of precalculation and lookup versus recalculation on the fly can be very different than for general purpose processors. For this application, for example, we found that precalculating the transmitter position, which is shared for all output pixels on a given pulse, did not significantly improve the execution time.

V. Performance Testing & Results

We tested the backpropagation implementation on a platform with two Intel Nehalem-based quad core processors operating at 2.4GHz, and two Tesla S1070. The test platform was Linux x64-based, and the test was built with the the CUDA 3.0 Toolkit. For our tests, we formed a 4400 x 4000 pixel output image from data sets containing 4000 pulses each. The input data for each image corresponded to sonar data collected over a period of approximately 60 seconds. The data used were collected by the SAS12 system, which was constructed under the sponsorship of the Office of Naval Research[4, 5]. We tested using from one to all of the eight available GPUs.

We created an 8 thread C-based implementation of the algorithm, but did not attempt significant optimization, including the use of SSE. This implementation used only single precision for all floating point operations. The C implementation was compiled with GCC 4.4.3, with –O4 optimization flag enabled. We also tested the formation of a single, representative 4400 long column of pixels near the center of the image using the MATLAB implementation in order to estimate its total runtime.

The measured runtimes, which include all data transfer to and from the GPUs, are summarized in Table 1, below. The reported flops/s was estimated by treating square roots, divisions, and trigonometric functions as single floating point operations. Under these assumptions, the minim flops per output pixel per input sample is 102 for SAS. Synthetic aperture radar is similar to SAS, but the much higher ratio of propagation speed to platform speed allows simplifications that reduce the load to 34 flops per pixel per pulse.

These results demonstrate a speedup of approximately 275x for our implementation relative to a lightly optimized, 8-core C based implementation, and significant enough speedup over the MATLAB implementation to make an otherwise time prohibitive technique available to radar and sonar algorithm researchers. We found that our implementation scaled fairly well up to 8 GPUs, achieving 72% linear speedup.

Table 1 - Execution Times

GPUs	Runtime(s)	flops/s
1	40.41	177.7E+9
2	21.08	340.6E+9
3	14.62	491.2E+9
4	11.35	632.7E+9
5	9.50	756.2E+9
6	8.29	866.0E+9
7	7.76	925.1E+9
8	7.02	1.0E+12
0 (8xCPU)	1931	3.70E+9
0 (Matlab)	*4320000*	*1.70E+6*

VI. Future Work

We expect to pursue several improvements to the SAS backpropagation module. While the implementation scaled up to 8 GPUs reasonably well, we expect that careful optimization of data transfer from host to GPU memory can further reduce runtimes. We also plan to add the ability to scale over multiple nodes in a distributed memory cluster in order to facilitate much larger image formation operations. Finally, we plan to encapsulate the implementation into a network service to allow resource pooling and support many algorithm researchers with a single platform.

VII. References

[1] H. J. Callow, *Signal Processing for Synthetic Aperture Sonar Image Enhancement*. PhD thesis, Department of Electrical and Electronic Engineering, University of Canterbury, Christchurch, New Zealand, 2003.

[2] D. A. Cook, "Synthetic Aperture Sonar Motion Estimation and Compensation," Master's Thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology, 2007.

[3] D. W. Hawkins, *Synthetic Aperture Imaging Algorithms: with application to wide bandwidth sonar*, PhD thesis, Department of Electrical and Electronic Engineering, University of Canterbury, Christchurch, New Zealand, 1996.

[4] A. D. Matthews, T. C. Montgomery, D. A. Cook, J. W. Oeschger, and J. S. Stroud, "12.75-inch synthetic aperture sonar (SAS), high resolution and automatic target recognition," in MTS/IEEE Oceans 2006 Boston, 2006.

[5] http://oceanexplorer.noaa.gov/explorations/08auvfest /logs /mam19/may19.html