GPU Accelerated, Sparse-Aware Ultrasonic Tomographic Imaging Using the Propagation and Back Propagation Method

Agustin Rivera-Longoria[1], Mark Idleman[2], Yuanwei Jin[3], Enyue Lu[4]

[1]Department of Computer Science, Texas State University, TX

[2]Department of Computer Science, Amherst College, MA

[3]Department of Engineering and Aviation Sciences, University of Maryland Eastern Shore, MD

[4]Department of Mathematics and Computer Science, Salisbury University, MD

INTRODUCTION

Nvidia's Compute Unified Device Architecture (CUDA) enables the parallel processing of algorithms on consumer-level graphics processing units (GPUs). CUDA uses a C-like programming model, giving developers a simple interface to utilize the parallel capabilities of modern day GPUs. Processes that need to be run in parallel are called on the CPU before being split up and run on the GPU; each parallel procedure is run on an individual thread, and threads are organized into groups called blocks. Data is transferred between the CPU and the GPU through memory copies and allocations. This means that CUDA programs can utilize both serial, CPU-hosted functions and GPU-hosted parallel functions. The architecture also provides a shared memory region for each block of threads, allowing for fast memory accesses across multiple threads in a parallel program.

The CUDA programming model allows the general user to utilize the capabilities of the GPU. The GPU's unique parallel capabilities can be used in a wide range of activities, including imaging, gaming, and computationally intensive applications.

PROBLEM DESCRIPTION

Medical imaging, gaming, and many other important real-world applications are computationally demanding processes due to the large number of quick calculations required at execution time. The introduction of GPUs has made it possible to parallelize these computations at a very low cost. In past work, we developed an iterative algorithm for nonlinear ultrasonic imaging using the propagation and backpropagation (PBP) method, and demonstrated that a GPU implementation of the algorithm could massively reduce the computation time.

In this work, we show that by incorporating target sparseness into the ultrasonic imaging algorithm, we can further accelerate the convergence rate of the algorithm by implementing it using a GPU.

The basis of the tomographic imaging problem starts with the following wave propagation equation:

$$\frac{\partial^2}{\partial t^2}u(x,t) = v^2(x)\Delta u(x,t) + \sum_{j=1}^{J^m} s(x, s_j, t)$$

where $u(x,t)$ is the acoustic field over $\Omega \times [0,T]$, J_m is the number of simultaneous excitation sources, $v(x) = \mu_0\sqrt{1+f(x)}$ is the acoustic wave propagation speed, and $f(x)$ is the acoustical potential function that we are reconstructing. The imaging problem can also be presented as an inverse problem

$$R_m(f) = g_m$$

where g_m is the data collected at the sensors surrounding the imaging field and R_m is a nonlinear operator. The solution to the sparsity-aware MIMO imaging problem is given by

$$c_\lambda^{(k+1)} = P_d(c_\lambda^k + \gamma^k)\left\langle R'_m(f^k)^*(g_m - R_m(f^k)), \varphi_\lambda \right\rangle$$

where $P_d(\cdot)$ is the projection operator, γ^k is the steepest descent coefficient, and $k = 0, 1, 2\ldots$ is the iteration timestamp.

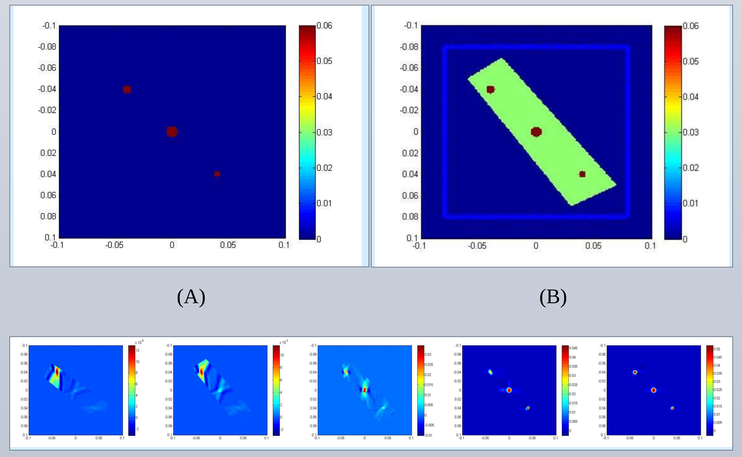
GPU IMPLEMENTATION

For our CUDA implementation of the sparsity-aware MIMO PBP imaging algorithm, we used the grid and block configuration that was found to have the best performance with the version of the algorithm that does not account for sparsity. In our implementation, each thread calculates a grid point (i.e. pixel) in parallel, lowering the execution time substantially.

By introducing a sparsity constraint into the algorithm, we faced a new parallel implementation issue, the race condition. Race conditions occur when multiple threads attempt to alter the same value in memory simultaneously, resulting in random and incorrect results. To successfully parallelize this type of calculation, we utilized parallel reduction, where partial sums are created in the shared memory of each thread block, and then added together after being copied back to the CPU to produce a final sum. Through the use of reduction, we achieved a speedup of approximately 4 seconds with our CUDA implementation.

RESULTS

We successfully implemented the sparsity-aware PBP imaging algorithm on the GPU. Our final results were benchmarked against both a MATLAB version of the algorithm and a C++ version. Our tests were run on a GeForce GTX 670 GPU, which has 1344 CUDA cores. We used a desktop computer with a quad core Xeon microprocessor running at 2.27 GHz and 8.0 GB of RAM. The setup of the imaging field and the iterative reconstruction process can be seen in Fig. 1.



(A) (B)

(C)

Figure 1: Examples of the imaging field prior to running the imaging algorithm and during the iterative reconstruction process. (A) The ground truth image. (B) The ground truth with prior knowledge of the target shown in green. (C) The reconstructed image during the first iteration of the algorithm. From left to right, the results after j = 10, 30, 100, 300, and 600, respectively, where j is the number of excitation sources that have propagated a wave across the imaging field during the first iteration.

RESULTS (CONTINUED)

In our tests, our CUDA implementation of the algorithm was approximately 460x faster than the MATLAB version and approximately 52x faster than the C++ version. The actual execution times and performance improvements can be seen in Table 1 and 2. Compared to a version of the imaging algorithm that does not account for image sparseness, our tests showed a performance improvement of 2.4x with the CUDA implementations of both algorithms (Table 3). This tangible performance improvement did not result in any degradation in the quality of the reconstructed image, however. The final result of the reconstructed image made using the sparsity-aware algorithm can be seen in Fig. 2.

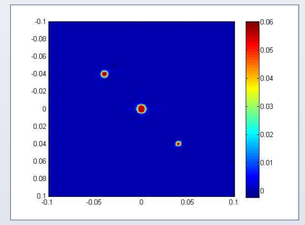


Figure 2: Final reconstructed image

	MATLAB	C++	CUDA
Time (hh:mm:ss)	05:52:38	00:39:53	00:00:46

Table 1: Processing times

Platform	MATLAB	C++	CUDA
MATLAB	1	0.113	0.002
C++	8.842	1	0.018
CUDA	459.957	52.020	1

Table 2: Performance tabulation

	Sparsity-Aware Algorithm	Base Algorithm
Time (hh:mm:ss)	00:00:46	00:01:50

Table 3: Performance of sparsity-aware algorithm vs. base algorithm

CONCLUSIONS

In this paper, we developed an effective GPU implementation of the sparsity-aware propagation and backpropagation imaging algorithm. The initial version of the algorithm, implemented in MATLAB, took just under six hours to complete. Our final CUDA implementation took an average of 46 seconds to execute, giving us an improvement of approximately 460x. When compared to the base imaging algorithm that doesn't account for image sparsity, we found an improvement of approximately 2.4x with our CUDA implementation, without compromising the quality of the reconstructed image.

The introduction of parallel sparse imaging calculations resulted in race conditions, which presented noticeable performance and accuracy problems. Parallel reduction was used to successfully alleviate race conditions, providing accurate results and improved performance.

ACKNOWLEDGEMENTS

The work is supported in part by the National Science Foundation under grant no. CCF- 1156509 and CMMI-1126008, and the U.S. Army Research Laboratory, the Office of Naval Research, and the Army Research Office under grant no. W911NF-11-1-0160.