



Efficient Numerical Algorithms on Graphics Hardware

Department of Computer Science

University of North Carolina at Chapel Hill

April 2007

Motivation

Numerical algorithms are an integral part of many scientific, data mining, multimedia and high performance computing applications. They have been extensively studied in the literature. These algorithms are designed to achieve high accuracy as well as high memory bandwidth and computational throughput on the CPUs. Many optimized numerical libraries such as ATLAS, Intel Math Kernel Library (MKL), FFTW are available and widely used for different applications. Many of these libraries have been optimized for different CPUs (e.g. Intel's MKL library is optimized for Intel processors).

Over the last few years, graphics processing units (GPUs) have become as ubiquitous as floating point processors. Current GPUs are programmable vector co-processors designed primarily for graphics and rasterization applications. Current GPUs can perform up to 10x more operations per second and can have up to 10x higher memory bandwidth than conventional CPUs. Moreover, the GPUs can effectively hide the memory latency in data-parallel applications. As a result, GPUs are also well-suited for numerical algorithms.

Our Approach

We give a brief survey of our recent work on developing novel algorithms to perform numerical computations on GPUs. Our algorithms use efficient data representations and pipeline the data processing instructions to different stages within a GPU. Within each stage of the graphics pipeline, we utilize the data parallelism to achieve higher computational throughput. Furthermore, our algorithms use tiling strategies to improve the GPU cache efficiency. Specifically, we present GPU-based numerical algorithms for:

- **LU and SVD Computations:** LU decomposition and SVD algorithms are basic components for solving linear systems of equations, eigen solvers and dimensionality

reduction techniques. Our algorithms proceed in multiple steps. In each step, the SVD algorithm [Bondhugula et al. 2006] applies a householder transformation while the LU-decomposition algorithm [Galoppo et al. 2005; Henson et al. 2005] involves multiply-and-add operations. Broadly speaking, these transformations are implemented using data-parallel vector operations on GPUs. For a matrix of size $n \times n$, our algorithms perform $O(n)$ transformations. Therefore, our LU and SVD routines require $O(n^3)$ memory accesses and computations. We represent the matrices using 2D texture representations on GPUs. We use simple fragment programs to efficiently perform the transformations on GPUs. Our algorithms are able to effectively pipeline the memory accesses and data processing. Furthermore, our algorithms are able to achieve high memory throughput on GPUs.

- **Fast Fourier Transforms:** Fast fourier transforms (FFTs) are fundamental for image-processing and signal-processing applications. FFT algorithms proceed in $\log n$ steps for a signal with n real or complex values. Each step performs $O(n)$ operations and the overall FFT algorithm requires $O(n \log n)$ operations. We have implemented the Stockham FFT of a large 1D signal by transforming the input signal into a 2D data representation [Govindaraju et al. 2006; Govindaraju and Manocha 2006a; Govindaraju and Manocha 2006b]. During each stage of the FFT, we decompose the input signal into data chunks with similar computations. The signal decomposition effectively reduces the number of instructions performed per data element. Next, we apply a simple fragment program to perform the operations in each step to the data elements. We further reduce the instructions by storing common computations among data elements in temporary 2D arrays. We also extend our 1D FFT algorithm to perform

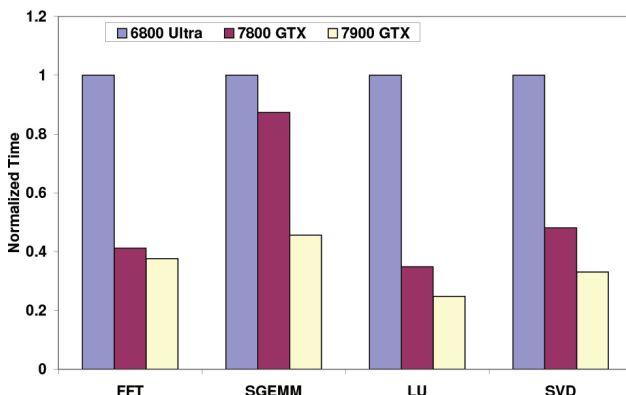


Figure 1: Performance of our numerical algorithms on three highend programmable GPUs released in 2004 (GeForce 6800), 2005 (GeForce 7800), and 2006 (GeForce 7900). We observe a performance improvement of 1.3 – 3x per year on different numerical applications.

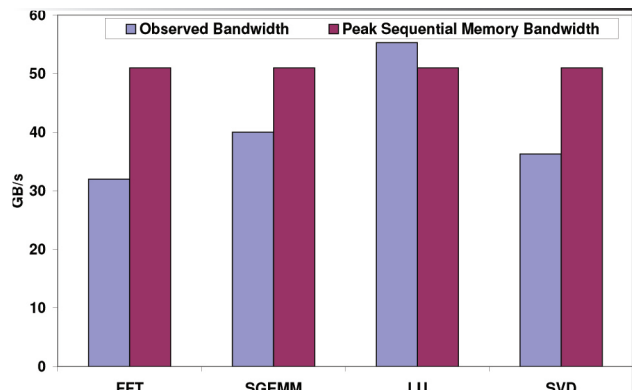


Figure 2: Observed memory performance of our numerical algorithms on a high-end NVIDIA 7900 GTX GPU and the peak sequential memory bandwidth on the 7900 GPU. The NVIDIA 7900 GPU has a peak sequential memory bandwidth of 51.2 GB/s. We observed a memory performance of 32–55 GB/s in our numerical algorithms.

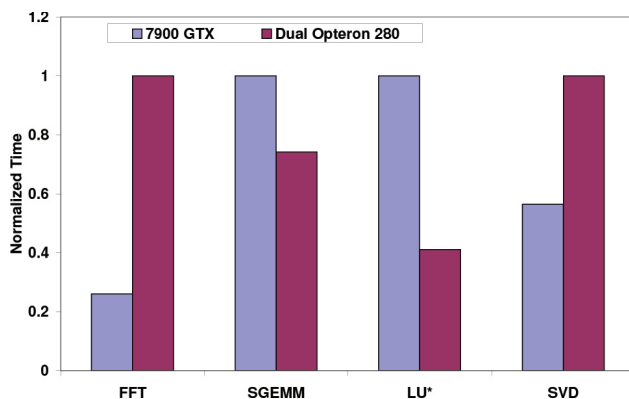


Figure 3: Normalized time spent in our numerical algorithms on a high-end NVIDIA 7900 GTX GPU costing \$500 and Intel's MKL library routines on a dual Opteron 280 processors costing over \$2000. Our experiments are conducted using single precision floating point matrices of size 2K 2K or 4 million complex 32-bit floating point values. Our CPU timings use all the four available processor cores for computation. Intel's MKL library only provides LU routines with partial and full pivoting; we compare the performance for partial pivoting.

FFTs of 2D signals. Our 2D FFT algorithm first applies the 1D FFT algorithm to the rows and then on the columns. Overall, our FFT algorithm is memory efficient and is able to achieve high computational performance on 1D real and complex single-precision signals. Moreover, the precision of our GPU-based FFT algorithm is comparable to single-precision FFT algorithms (using IEEE 32-bit arithmetic) on CPUs. In practice, we have observed that the precision is close to 1.5×10^{-6} for 1D FFTs with tens of millions of values.

- **Dense Matrix Multiplication:** The problem of matrix multiplication is inherently parallel and memory intensive – therefore, it can greatly benefit from the high computational throughput and memory performance of GPUs. Our matrix multiplication algorithm uses pipelining strategies to map the computation to different stages of the graphics pipeline. We further enhance the performance of our algorithm using a modified blocking strategy based on our memory model for GPU computations [Govindaraju et al. 2006; Govindaraju and Manocha 2006a]. Our cache-optimized algorithm requires $O(n^3)$ compute operations for multiplying $n \times n$ matrices and reduces the sequential memory accesses to $O(n^3 + n^3/D)$ where D is a blocking parameter.
- **Sparse Matrix Vector Multiplication (SpMV):** SpMV is a widely used kernel in various applications of scientific computation and engineering. SpMV is notoriously known for indirect and irregular memory accesses due to poor temporal and spatial locality. We are implementing algorithms for parallelizing this computation on many cores and optimizing memory hierarchy using Block Compressed Sparse Row (BCSR) representation. We are using CUDA on NVIDIA G80. Our current implementation achieves 3.37 GFLOPS with 40.32 GB/s memory bandwidth.

We have implemented our algorithms using OpenGL and tested the performance on a commodity PC with different GPUs, including NVIDIA GeForce 6800, 7800 and 7900. In practice, our algorithms are able to achieve 3–17G single-

precision FLOPS of performance on the different applications. Fig. 3 highlights the computational performance of our GPU-based algorithms and compare them to Intel math kernel library implementation on a high-end PC with dual AMD Opteron 280 processors. In our CPU implementation, we use all four threads to utilize all the available processors. In practice, the performance of our GPU-based algorithms is comparable to optimized CPU-based algorithms. Fig. 2 shows the measured memory performance of our algorithms and the peak sequential memory performance. Our algorithms are able to achieve 32–55 GB/s of measured memory bandwidth. Fig. 1 highlights the performance of our algorithms on three high-end NVIDIA GPUs, released in successive generations. We observe a 1:3–3x performance improvement per year.

Team Members

Dinesh Manocha, Phi Delta Theta/Matthew Mason
Distinguished Professor

Naga K. Govindaraju, assistant research professor (now at Microsoft Research)

Nico Galoppo, graduate research assistant

Michael Henson, graduate research assistant

Vinay Bondhugula, graduate research assistant

Scott Larsen, graduate research assistant

Sashi Kumar Penta, graduate research assistant

Research Sponsors

U.S. Army Research Office

Disruptive Technology Office

National Science Foundation

Defence Advanced Research Projects Agency

RDECOM

References

Bondhugula, V., Govindaraju, N. K., and Manocha, D. "Fast singular value decomposition on graphics processors," Technical report, University of North Carolina at Chapel Hill, 2006.

Galoppo, N., Govindaraju, N., Henson, M., and Manocha, D. "Lu-gpu: Efficient algorithms for solving dense linear systems on graphics hardware," *Supercomputing, 2005. Proceedings of ACM/IEEE SC 2005 Conference*. 2005.

Govindaraju, N. K., and Manocha, D. "Cache efficient numerical algorithms using graphics hardware," Technical Report, University of North Carolina at Chapel Hill, 2006.

Govindaraju, N. K., and Manocha, D. "Gpufftw: High performance power-of-two fft library using graphics processors," available online: <http://gamma.cs.unc.edu/GPUFFTW>. 2006.

Govindaraju, N. K., Larsen, S., Gray, J., and Manocha, D. "An efficient memory model for scientific algorithms on graphics processors," Technical report, University of North Carolina at Chapel Hill, 2006.

Henson, M., Galoppo, N., Govindaraju, N., and Manocha, D. "Lugpu: Library for solving dense linear systems on graphics hardware," available online: <http://gamma.cs.unc.edu/LUGPULIB>. 2005.