CUDA Based GPU Solvers For Method of Moment Simulations

Matthew J. Inman\textsuperscript{1}, Atef Z. Elsherbeni\textsuperscript{1}, and C. J. Reddy\textsuperscript{2}

\textsuperscript{1}Department of Electrical Engineering
University of Mississippi, University, MS 38677-1848, USA
atelyf@olemiss.edu, mjinman@olemiss.edu

\textsuperscript{2}Applied EM
Hampton, VA 23666, USA

\textbf{ABSTRACT:} The use of Graphical Processing Units (GPU’s) to perform computational tasks required by electromagnetic simulations have been shown over the past several years to increase the computational speed significantly [1-2]. Current generation of GPU’s can contain up to 240 computational cores with a memory bandwidth an order of magnitude faster than the CPU. This hardware coupled with NVIDIA’s CUDA allow for programs to execute certain subroutines directly on the GPU. By incorporating CUDA subroutines in a complex double-precision LU decomposition solver, it will be shown that the massively large memory bandwidth and number of arithmetic logic of units the GPU will significantly increase the computational speed in relation to solving dense matrix equations such as those found in method of moment simulations.

\textbf{Keywords:} GPU, MOM, Dense Matrices, LU Decomposition

1. Introduction

As computational power has increased exponentially over the past few decades, the need for solving complex systems of equations has grown equally in tandem. Even in method of moment simulations, simple geometries can often lead to complex matrices whose size can order in the thousands. In order to accurately and quickly solve these simulations, especially cases where there are many of right hand sides to be calculated, an appropriate solution method must be chosen. This paper will address the use of GPU based LU decomposition solvers for the matrix solutions.

LU decomposition offers many advantages over other decomposition, inversion, and direct solution solvers as it applies to the GPU. For a large number of right hand sides, direct solution solvers become unwieldy to implement as each right hand side requires its own solution. Inversion methods can allow for the solving at will after the matrix has been inverted but often require large computational runtimes and can suffer from instability as the order of the matrix becomes too large. Decomposition methods offer a good compromise between full inversion and direct solution. LU decomposition in particular lends itself well to implementation on the GPU.

Many of the computations required for LU decomposition can be offloaded to the GPU by using CUDA. While LU decomposition on the GPU has previously been demonstrated to outperform the CPU, the past work has been limited to only solving real matrices. For LU decomposition to be of use in computational electromagnetics any implementation must include support for complex numbers.
In the construction of a complex double-precision LU decomposition solver on the GPU, several objectives must be met. While many of the subroutines used in LU decomposition can be run on the GPU faster than the CPU, some portions of the code are still more appropriate to run on the CPU. Maintaining data integrity between CPU and GPU complex double-precision data types must be preserved. The inclusion of double-precision calculations will also be examined from a memory standpoint in optimizing the local cache memory in the GPU for the fastest execution times.

2. Complex Double-Precision LU Decomposition in CUDA

LU Decomposition has been previously demonstrated on the GPU using CUDA for single precision real matrices demonstrating speed gains approaching an order of magnitude over common CPU’s [3-4]. These solvers mixed a combination of CPU BLAS calls with CUDA CUBLAS (NVIDIA’s GPU based BLAS libraries) calls and a few minor CUDA kernels to facilitate the process. The demonstrated solver was easily facilitated by the near complete status of CUBLAS for single precision real data types. While these solvers showed decent speed, the restriction of single precision real data types limits its usefulness in electromagnetic simulations. In order to solve most common CEM problems a solver capable of working with double precision complex numbers is necessary.

Double precision complex development on the CUDA platform presents several unique challenges to be addressed. These challenges occur from the incomplete development of the CUBLAS libraries. Currently the CUBLAS libraries only support complete BLAS routines in single precision real and only very limited support for single and double precision complex. In the current version of CUBLAS only 2 out of 13 level 1 BLAS routines, 1 out of 16 level 2 BLAS routines, and 2 out of 6 level 3 BLAS routines are supported. In the development of a single precision code for LU decomposition, the CUBLAS libraries can be extensively used. For double precision code with support for complex numbers, the CUBLAS libraries must be supplemented with custom CUDA kernels and CPU based BLAS routines.

In order to compensate for the lack of several appropriate BLAS routines in CUBLAS, the “Zstrsm” function has been offloaded to the CPU, while the transpose functions has been written in CUDA with support for double precision complex numbers. Offloading the “Zstrm” function back to the CPU also presents problems in maintaining data consistency. When working with single or double precision real numbers, transferring data between CUBLAS and Intel MKL BLAS (the CPU BLAS used here) is trivial as these routines operate with the same data types (float or double). In complex, however, MKL BLAS and CUBLAS have different data types and data structures to represent the data. In order to accomplish consistent data transfer the MKL BLAS has been modified so that its data structure is compatible with CUBLAS data types. To use the MKL BLAS functions the CUBLAS data types must be forced recast into MKL BLAS.

The custom routines written in CUDA for transposition were crafted to support the complex double precision numbers. The added data overhead requires smaller blocks of the matrix to be transferred at a single time (since 1 element of a double precision complex matrix has 4 times the data as a single precision real matrix). The transpose routines make use of local GPU cache memory in order to make this process as fast as possible. At this point these routines have only begun to be optimized for speed as the memory required for complex double precision as well as the memory layout makes this process difficult.

3. Example Problem Definition

To show the use of the GPU based solver, a well known sample problem was chosen. In this sample, the current along a wire antenna of length \( L \) (0.1m) and radius \( A \) (0.1mm) that is excited by a magnetic
frill model will be calculated as shown in figure 1. This simulation will be calculated using sinusoidal basis functions and mid-point integration.

Figure 1. Sample wire antenna configuration.

The sample problem was chosen in order to validate the simulations against existing codes and for its simplicity in integration into the GPU solver codes. Because of its nature it is simple to change the discretization of problem and examine the solution times as a function of the subsequent matrix size.

4. Results

The GPU code was run against the reference codes to ensure proper operation. Figure 2 shows the current along the wire in both codes for a sample discretization of 1024 segments. The results show very good agreement with only very minor differences in the magnitude of the current. These differences (less than 0.1%) can be attributed to minor differences in how the numbers were stored and calculated in the various programs and the use of the GPU in the simulation. The errors in the phase calculations were even smaller by several degrees of magnitude which means the differences were most likely due to the differences in how the GPU and CPU handles rounding.

Figure 3 shows the various solution times for different matrix sizes. These solution times were measure using the same program operating in either CPU only mode (using Intel MKL BLAS for the calculations) or in CPU+GPU mode (Using NVIDIA CUBLAS to operate on the majority of the simulation). These are the simulation only run times and do not include matrix fill times. The results shown are for several different configurations of systems and graphics cards as noted on the figures. From these results it can be seen that as the matrix size increases the GPU codes run approximately twice as fast as the CPU only codes run compared to a quad core 2.6 GHz Intel i7 machine and approximately 4 times faster compared to a 2.4 GHz Intel Core Duo. Since the GPU only has begun to support double precision calculations recently this slow down can be attributed to the immaturity of the GPU hardware.

Figure 2. CPU and reference results for current and phase.
The introduction of double precision support on the GPU has allowed for its use in a wide range of CEM solvers. It has been shown here that even a moderate speed gain of 2 can be achieved in the dense matrix solving in a Method of Moments problem over a high end PC. The solutions show good agreement with reference codes and can be implemented with relative ease as a replacement for CPU only LU decomposition solving. Future work can be done to further optimize the codes by replacement of certain CUBLAS routines in double precision and the matrix fill can also be studied for implementation on GPU.

References