CUDA Based LU Decomposition Solvers for CEM Applications

Matthew J. Inman[1], Atef Z. Elsherbeni[1], and C. J. Reddy[2]

[1] Department of Electrical Engineering
University of Mississippi, University, MS 38677-1848, USA
atef@olemiss.edu , mjinman@olemiss.edu

[2] Applied EM
Hampton, VA 23666, USA
cjreddy@emssusa.com

Abstract ─ The use of graphical processing units to perform numerical computations required by electromagnetic analyses have been shown over the past several years significant increase in the computational speed. Most of the previous work concentrated on electromagnetic analyses that do not require matrix inversion. This paper uses the NVIDIA's compute unified device architecture (CUDA) language to develop and modify routines for matrix solution based on the LU decomposition procedure to enhance and speed up a class of electromagnetic simulations. This implementation is utilizing the CPU and GPU for the inversion procedure. Various implementations for real, complex, single precision and double precision will be examined. The performance details of the developed LU decomposition routines especially for complex and double precision arithmetic are presented.

Index Terms ─ CUDA, GPU, CEM, LU Decomposition, Matrix Solvers.

I. 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 simple geometries can often lead to complex matrices whose size can easily be in the order of thousands. In order to accurately and quickly provide results from these simulations an appropriate solution method must be chosen. This paper will address the use of graphical processing units (GPU's) based LU decomposition solvers for matrix solutions.

The LU decomposition offers many advantages for solving dense matrices. Full inversion methods (such as Gaussian elimination) can allow for the solving of many right hand sides easily once the inversion is complete. However, full inversions often require large computational runtimes compared with LU decomposition. Many parts of LU decomposition lend itself well to implementation on the GPU due to its past widespread use on other various parallel computing systems [2-6].

Using the NVIDIA compute unified device architecture (CUDA) interface, many of the computations required for LU decomposition can be offloaded to the GPU. While LU decomposition on the GPU has previously been demonstrated to outperform the CPU [3-4], past published work has been mainly limited to real matrices in single precision. In order for LU decomposition to be of widespread use in computational electromagnetics (CEM), any GPU implementation must be able to support complex values. Large matrixes will also require double precision support in order to maintain stability.

In this paper the construction of LU decomposition solver on the GPU is performed using existing and newly developed routines. While many of the subroutines used in LU

decomposition can run on the GPU faster than the CPU, some portions of the code are still more appropriate to run on the CPU [2-3]. Maintaining data integrity between CPU and GPU for complex double precision numbers must be established. The inclusion of double-precision calculations will also be examined from a memory standpoint in optimizing the local cache memory in the GPU to achieve the fastest execution possible.

II. Data Types and Computations Efficiency

It is widely known that different data types can have a large effect on the computational runtime for any algorithm. For instance, going from any real data type to a complex one not only doubles the amount of memory required to move and store, but the complexity of even simple arithmetic operation increases by a significant amount. Complex addition and subtraction requires two separate additions or subtractions. Multiplication requires one addition, one subtraction, and four multiplications. Division requires eight multiplications, three additions, one subtraction, and two divisions. This increase in complexity for complex numbers can have major effects on the runtime of any algorithm.

In addition, the change from single to double precision calculations can have a likewise effect on performance. The double precision performance of the NVIDIA Tesla C1060 is almost 12 times slower than single precision. An Intel Core i7 CPU has double precision speed only 1.4 times slower. This major discrepancy is due to both the maturity of the arithmetic hardware and how this hardware is implemented. Double precision support on NVIDIA GPU's are only a single generation old and are implemented by combining multiple single precision units together to create a double precision unit. Future generation of NVIDIA Fermi GPU's are expected to have better double precision support according to the vendor information that are about to be released.

In this paper we will consider various aspects in the comparison between solvers utilizing different data types. The amount of data to be transferred and stored in system, the increase in computations required for complex number, and the efficiency of double precision calculations will be taken into account. Comparisons will address all these issues within the results.

III. LU Decomposition Solvers in CUDA

The LU decomposition has been previously demonstrated on the GPU using CUDA and other programming techniques for single precision real matrices [3-4]. Published result produced speed gains approaching an order of magnitude over common CPU's. These solvers mixed a combination of CPU Basic Linear Algebra Subprograms (BLAS) calls, CUDA CUBLAS (NVIDIA's GPU based BLAS libraries) calls, and CUDA kernel. The BLAS libraries contain highly tuned functions commonly used in many programs to perform basic linear algebra. The published LU solvers were facilitated by the complete and mature development of CUBLAS libraries for single precision real data types. These solvers showed a speed increase of 6 to 12 times (relative to various hardware). However, the restriction of single precision real data types limits its usefulness for CEM simulations. Many common CEM problems require the solver to be available for any combination of single precision, double precision, real, and complex data.

The development of solvers that support data other than a real single precision on the CUDA/GPU platform presents several unique challenges to be addressed. These challenges occur from the status of the CUBLAS libraries. The CUBLAS libraries (previous to release 3.0) only supported complete BLAS routines in single precision real and only very limited support for single and double precision complex. In the utilized version 2.0 of CUBLAS for this paper, 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 were supported. The CUBLAS version 3.0, recently released, claims full support for all BLAS routines in all data types.

With the release of CUBLAS 3.0 it is now possible to perform the LU decomposition directly on the GPU without the aid of any CPU calls.

However, this does not mean that the CUBLAS functions outperform their CPU based counterparts. Certain linear algebra functions still perform significantly faster (such as factorization) on the CPU compared to the GPU's as utilized in this paper. The algorithm presented here was carefully profiled to determine when and which parts of the LU decomposition routine can be solved on the GPU with maximum efficiency.

The real single precision solver presented here follows the published methodology of utilizing both the CPU and the GPU as in [3-4] and the established algorithms for parallel computing systems [5]. The code has been programmed and tuned by the authors using these methods. In order to extend this solver for other data types, some of the CUBLAS calls have been replaced with custom developed kernels (GPU functions).

In the solvers presented here, the "*trsm" function which is a standard BLAS routine used to solve a triangular matrix, has been offloaded to the CPU. The transpose functions have been developed in CUDA to support all types of data (complex and real in single and double precisions). With this added support for the various data types, the developed GPU code was tuned for various block sizes which determines how much data gets transferred, at a time, between the GPU and CPU. Offloading the "*trsm" function back to the CPU also presents problems in maintaining data consistency. The transfer of data between CUBLAS on GPU and Intel MKL BLAS on CPU is simple when working with single (float) or double precision real numbers. However, for complex data, MKL BLAS and CUBLAS have different data types and data structures to represent the numbers. In order to accomplish consistent data transfer, the MKL BLAS has been modified so that its data structure is compatible with CUBLAS data types. This modification allowed the free exchange of data between CUBLAS on the GPU and MKL BLAS on the CPU for complex numbers.

The custom routines in CUDA for transposition and pivoting were developed to support all combinations of data types. Depending on the data type needed, the additional data overhead requires smaller blocks of the matrix to be transferred at a single time (as a double precision complex matrix has 4 times the data as a single precision real matrix). The transpose routines make use of local cache memory inside the GPU in order to make this process as efficient as possible.

Table 1 details the various functions used for the developed CPU+GPU based LU decomposition and where they are performed. The basic algorithm iterates through the various block columns of the matrix and performs the decomposition as detailed in [5]. Each block is first transposed and the L/U matrices are updated on the GPU. The block is transferred to the computer system and factorization takes place on the CPU. The block then streams through the GPU for pivoting and back to the CPU. The block is then inverted and the L matrix is solved. The update for the U matrix is performed on the GPU, then the data is transferred back and the final U solve is done on the CPU.

Table 1: Functions required for LU decomposition

Transpose Block	GPU (CUDA Kernel)
Matrix Multiply	GPU (CUBLAS)
Factorization	CPU (MKL BLAS)
Pivot	GPU (CUDA Kernel)
Triangular Matrix Solve	CPU (MKL BLAS)

Each of the functions listed in Table 1 can be implemented on either the CPU or the GPU. For the factorization and the matrix solve routines, the CPU was more efficient in processing even with the added overhead of transferring the data. Both of these functions are not easily parallelized which explains why they are more efficiently performed on the CPU. The transpose and pivoting functions were written in CUDA and optimized for each data type and block size. This necessitated writing separate CUDA functions for each separate data type in order to maintain the highest processing speed possible.

IV. LU Solver Results

The developed CUDA based LU solver was implemented on different systems for various data types. Similarly a pure CPU solver based on the Intel MKL library was used for all comparisons. The solvers were run on various CPU and CPU+GPU based configurations as detailed in Table 2. In all cases, the Intel MKL library uses all available cores on a CPU (2 cores on Core Duo, and 4 cores on i7).

Table 2: System configurations

System	
System 1	2.66 GHz Intel Core i7 6GB DDR3 PC12800 NVIDIA 280GTX 1GB NVIDIA Tesla C1060 4GB
System 2	2.4 GHz Intel Core Duo 4GB DDR2 PC4700 NVIDIA 8800GTX 768 MB

Figure 1 shows the runtime results for the first case of single precision real data for CPU and CPU+GPU implementations on various systems. This baseline case matches other published results [3] in runtimes and speed gain. The CPU+GPU implementations outperformed the CPU only implementation anywhere from 3 to 12 times based on the configuration of the CPU and GPU.

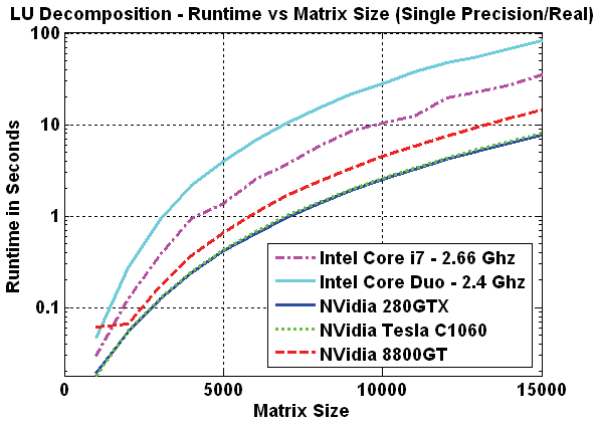

Fig. 1. Runtimes for real single precision LU decomposition.

In the real single precision case, the implementation is quite simple and the best speed gain can be realized. When the solver is expanded to double precision, the results show a moderate decrease in speed for all the available cases as seen in Figure 2. Only the NVIDIA 280 and Tesla C1060 support GPU based double precision and thus are shown here. The Intel Core i7 is the CPU for both the CPU and CPU+GPU cases in this figure. For this real double precision case, the CPU only implementation increased the runtime speed by roughly double across all the various matrix sizes, while the CPU+GPU implementation increased runtime by only around 90% over the single precision case.

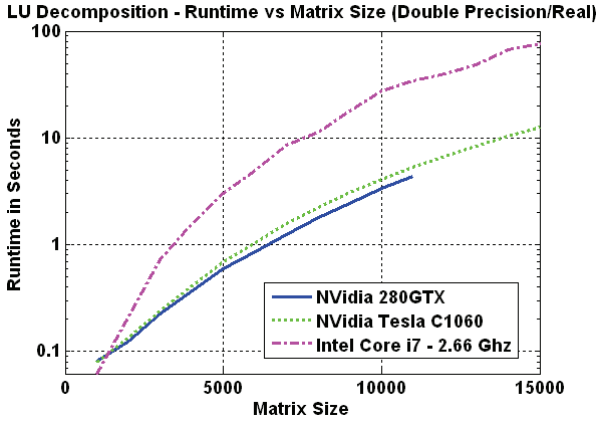

Fig. 2. Runtimes for real double precision LU decomposition.

In the real double precision cases, the CPU+GPU implementation achieved a speed gain of seven times over the CPU only based counterpart. Interestingly, even though twice the amount of data is required to be moved for a double precision case and known inefficiencies of the GPU processing double precision data, the CPU+GPU case only increased runtime by 90%. This can be explained by examining the memory access patterns in processing double precision data. In algorithms such as LU decomposition, data access to the memory of the CPU and GPU are not optimal for the fastest transfer. The addition of double precision data in these cases actually increase the efficiency of memory access since larger blocks of linear memory is being read at a single time. The addition of double precision arithmetic for these cases did not account for any

noticeable increase in processing time. This is due to the fact that in these cases the arithmetic is fairly simple. The calculations were completed before the next block of data has arrived from memory even with the overhead of double precision calculations.

The last implementation presented is the complex double precision case. Figure 3 shows the runtimes for various configurations. With the addition of complex numbers, the runtimes have slowed significantly over the real single precision cases. The CPU only implementation runs approximately 9 times slower while the CPU+GPU implementation runs approximately twenty times slower. It can still be seen that in all cases the CPU+GPU implementation still outperforms the CPU only implementation by approximately two (2.66 Intel Core i7) to four (2.4 GHz Core Duo).

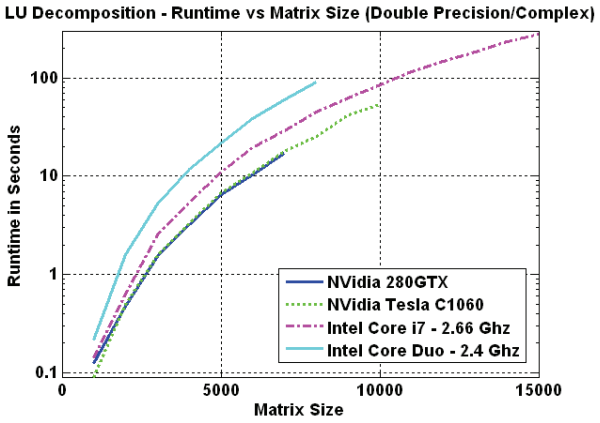

Fig. 3. Runtimes for complex double precision LU decomposition.

The addition of complex data to the solver showed a drastic effect on the runtimes of the developed LU solvers. In order to understand how the various implementations performed, it is necessary to examine how the CPU only and the CPU+GPU implementations compared against themselves. Figure 4 shows the runtimes on the Intel Core i7 for the CPU only implementations.

The addition of double precision to the implementation increased the runtime by only double. Since twice the data is being transferred in this case, it can be concluded that for the real single and double precision cases, the runtimes are simply a matter of the memory transfer rates. In the complex double precision case, the runtimes lagged the real single precision case by a factor of approximately 7. Since four times the data is required to be transferred it can be seen that the arithmetic itself becomes the limiting factor in performance.

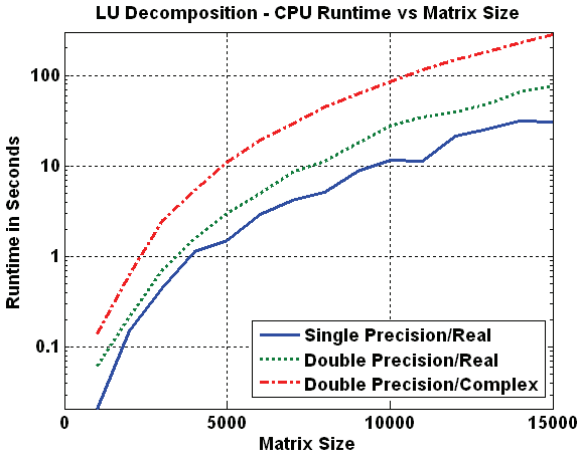

Fig. 4. Runtimes for CPU only LU decomposition.

Figure 5 shows the comparison for the CPU+GPU cases running on the Intel Core i7 with the NVIDIA Tesla C1060 GPU. As shown before, the double precision increased the runtime relative to the single precision by only around 90%. While the data transferred did double, the GPU was able to handle the data more efficiently and thus did not require twice the time to make the transfer. Likewise from the CPU only cases, the memory transfer rates appear to be the limiting factor in the runtimes for these cases. However, in the complex double precision case, the slowdown is more pronounced. The runtime for the complex double precision is approximately twenty times slower over the real single precision case. Just as with the CPU only case, the complex double precision implementation becomes limited not by the memory access rate, but by the speed the system can perform the computations. Since the current CUBLAS on GPU is nowhere near as efficient as the MKL BLAS on CPU in performing double precision calculations, the GPU performance suffers a larger runtime penalty.

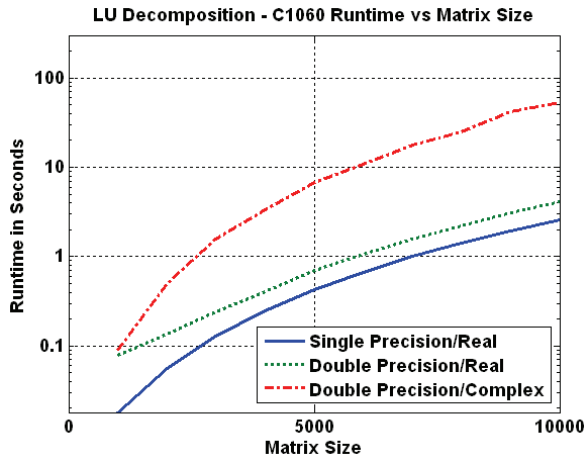

Fig. 5. CPU+GPU LU decomposition runtimes.

VI. Verification and Examples

To show the advantage of the CPU+GPU based solver, few examples were tested. These examples are based on a method of moments (MoM) solution whose results are well documented. Each of these examples will be used to compare both the speed and the accuracy of the CPU+GPU based solutions relative to the CPU only solution. For simplicity, all examples will be discretized with 4096 segments and run on an Intel Core i7 with a NVIDIA 280GTX. The 4096 segments were choosen to show the performance for a simulation of decent size. The CPU only code utilizes all 4 cores of the Core i7 and the CPU+GPU code utilizes the same with the addition of the graphics card. All solutions were computed with double precision complex solvers.

The first example is a simple wire dipole antenna. This example will calculate the current along a wire antenna of length ***L*** (0.1m) and diameter ***A*** (0.2mm) that is excited by a magnetic frill model as shown in Fig. 6. Sinusoidal basis functions and mid-point integration procedure are used for the solution of the resulting integral equation.

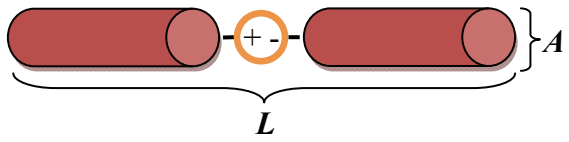

Fig. 6. Dipole wire antenna configuration.

The CPU+GPU code was run against the reference codes to ensure proper operation. Figure 7 shows the current along the wire in both codes. The results show very good agreement with only very minor differences in the magnitude of the current. These differences which are 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.

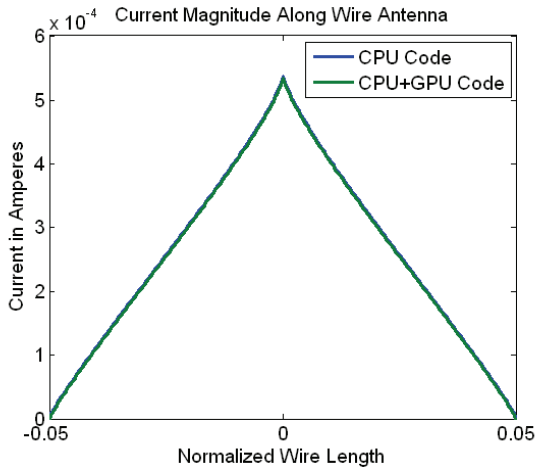

Fig. 7. Current distribution along the dipole wire antenna.

The second example shows the calculation of the current distribution along a PEC plate illuminated by a TM_z plane wave. Figure 8 shows the configuration of this setup. In this setup the width of the PEC plate is one wavelength and the TM_z plane wave incident to the face of the plate at a 45 degree angle.

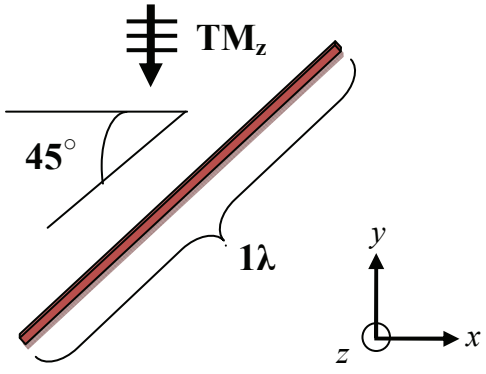

Fig. 8. PEC plate and excitation configuration.

This example was run and compared against the reference code as seen in Fig. 9. The CPU+GPU code again show excellent agreement in calculating the surface current of the PEC plate. The maximum error observed between the two solvers is 0.08%.

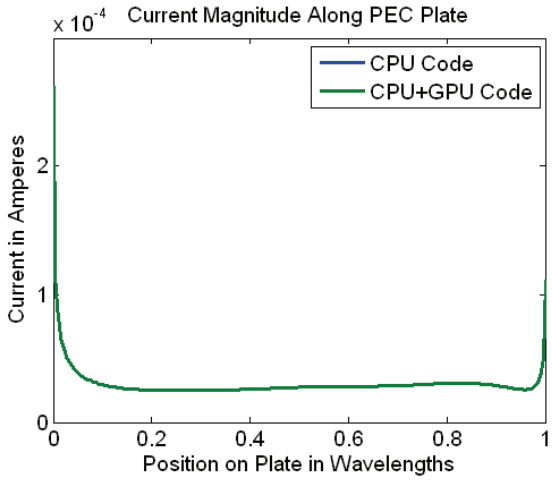

Fig. 9. Current distribution along the PEC plate.

The last example shows the calculation of the current distribution along a PEC cylinder illuminated by a TM_z plane wave. Figure 10 shows the configuration of this setup with the diameter of the PEC cylinder being one wavelength.

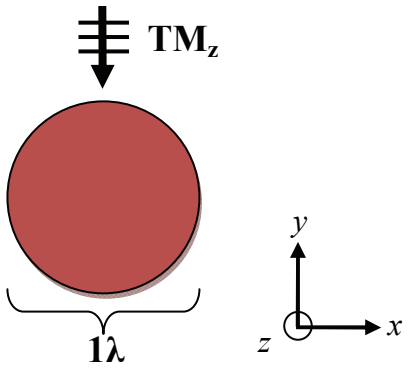

Fig. 10. PEC cylinder and excitation configuration.

Figure 11 shows the current magnitude along the PEC cylinder for both cases. As shown, the agreement between the two codes is excellent. In this case, the maximum error between the CPU and CPU+GPU codes was less than 0.02%

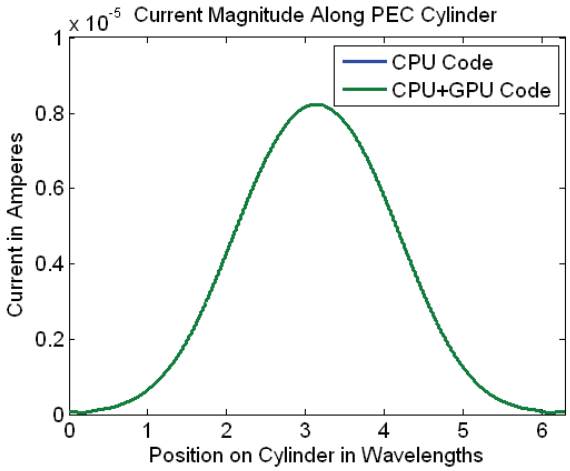

Fig. 11. Current distribution along the PEC cylinder.

All three of the sample cases show excellent agreement with the CPU only solver and successfully solved the problems utilizing the GPU. For these cases a single solve time on the CPU required approximately 6.3 seconds while the CPU+GPU only required 3.2 seconds. Many cases in computational electromagnetics, such as computing the monostatic RCS of an object, require solving for hundreds or more of right hand sides. The speed increase shown for even a moderate matrix of rank 4096 can halve the solution time compared against a high end CPU. If double precision is not required, the time savings can be even greater.

VI. Conclusions

It has been shown that an LU decomposition solver can be effectively implemented utilizing the GPU for various data types from real single precision to complex double precision. Due to the nature of certain functions required for LU decomposition, the use of the CPU to perform various operations is necessitated.

While the complex double precision LU decomposition solver did not maintain increase in speed as for the real precision cases did, the increase of two-fold can have a drastic effect on CEM simulation times, especially for problems of multiple right-hand sides. The decrease in speed gain from the CPU+GPU implementation in the

complex double precision cases can be easily attributed to the immature state of double precision arithmetic on this generation of GPU's. Future generations of GPU's have been promised to dramatically increase double precision arithmetic computations speed which should allow for greater utilization of the developed GPU routines for faster solutions to a variety of CEM and other applications.

REFERENCES

[1] M. J. Inman and A. Z. Elsherbeni, "Programming video cards for computational electromagnetics applications," *IEEE Antennas Propagation Mag.*, Vol. 47, Issue 6, pp. 71-78, 2005.

[2] K. Fatahalian, et. al., "Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication", Stanford University, 2004.

[3] V. Volkov and J. W. Demmel, *Benchmarking GPUs to tune dense linear algebra*, SC08, 2008

[4] N. Galoppo, N. Govindaraju, M. Henson, and D. Manocha, *LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware*, Proceedings of the ACM/IEEE conference on Supercomputing, 2005.

[5] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. Mckenney, J. Du Croz, S. Hammerling, J. Demmel, C. Bischof, And D. Sorensen, LAPACK: a portable linear algebra library for high-performance computers, *Supercomputing '90*, 1990.

[6] M. Baboulin, J. Dongarra, and S. Tomov. Some Issues in Dense Linear Algebra for Multicore and Special Purpose Architectures, LAPACK Working Note 200, 1993.

[7] CUDA User Forums, http://forums.nvidia.com

Matthew Joseph Inman received his B.S. in Electrical Engineering in 2000 and his Masters in Electromagnetics in 2003 from the University of Mississippi. He is currently pursuing Ph. D. studies in electromagnetics there. He is employed by the University of Mississippi as a research assistant and graduate instructor teaching a number of undergraduate courses. His interests involve electromagnetic theories, numerical techniques, antenna design and visualization.

Atef Z. Elsherbeni is a Professor of Electrical Engineering and Associate Dean for Research and Graduate Programs, the Director of The School of Engineering CAD Lab, and the Associate Director of The Center for Applied Electromagnetic Systems Research (CAESR) at The University of Mississippi. In 2004 he was appointed as an adjunct Professor, at The Department of Electrical Engineering and Computer Science of the L.C. Smith College of Engineering and Computer Science at Syracuse University. On 2009 he was selected as Finland Distinguished Professor by the Academy of Finland and Tekes. Dr. Elsherbeni has conducted research dealing with scattering and diffraction by dielectric and metal objects, finite difference time domain analysis of passive and active microwave devices including planar transmission lines, field visualization and software development for EM education, interactions of electromagnetic waves with human body, sensors development for monitoring soil moisture, airports noise levels, air quality including haze and humidity, reflector and printed antennas and antenna arrays for radars, UAV, and personal communication systems, antennas for wideband applications, antenna and material properties measurements, and hardware and software acceleration of computational techniques for electromagentics. Dr. Elsherbeni is the co-author of the book "*The Finite Difference Time Domain Method for Electromagnetics With MATLAB Simulations"*, SciTech 2009, the book "*Antenna Design and Visualization Using Matlab*", SciTech, 2006, the book "*MATLAB Simulations for Radar Systems Design*", CRC Press, 2003, the book *"Electromagnetic Scattering Using the Iterative Multiregion Technique"*, Morgan & Claypool, 2007, the book *"Electromagnetics and Antenna Optimization using Taguchi's Method"*, Morgan & Claypool, 2007, and the main author of the chapters "*Handheld Antennas"* and "*The Finite Difference Time Domain Technique for Microstrip Antennas"* in Handbook of Antennas in Wireless Communications, CRC Press, 2001. Dr. Elsherbeni is a Fellow member of the Institute of Electrical and Electronics Engineers (IEEE) and a Fellow member of The Applied Computational

Electromagnetics Society (ACES). He is the Editor-in-Chief for ACES Journal and an Associate Editor to the Radio Science Journal.

C. J. Reddy received B. Tech. degree in Electronics and Communications Engineering from Regional Engineering College (now National Institute of Technology), Warangal, India in 1983. He received his M.Tech. degree in Microwave and Optical Communication Engineering and Ph.D. degree in Electrical Engineering, both from Indian Institute of Technology, Kharagpur, India, in 1986 and 1988 respectively. From 1987 to 1991, he worked as a Scientific Officer at SAMEER (India) and participated in radar system design and development. In 1991, he was awarded NSERC Visiting Fellowship to conduct research at Communications Research Center, Ottawa, Canada. Later in 1993, he was awarded a National Research Council (USA)'s Research Associateship to conduct research in computational electromagnetics at NASA Langley Research Center, Hampton, Virginia. Dr. Reddy worked as a Research Professor at Hampton University from 1995 to 2000, while conducting research at NASA Langley Research Center. During this time, he developed various FEM codes for electromagnetics. He also worked on design and simulation of antennas for automobiles and aircraft structures. Particularly development of his hybrid Finite Element Method/Method of Moments/Geometrical Theory of Diffraction code for cavity backed aperture antenna analysis received Certificate of Recognition from NASA.

Currently, Dr. Reddy is the President and Chief Technical Officer of Applied EM Inc, a small company specializing in computational electromagnetics, antenna design and development. At Applied EM, Dr. Reddy successfully led many Small Business Innovative Research (SBIR) projects from the US Department of Defense (DoD). Some of the technologies developed under these projects are being considered for transition to the DoD. Dr. Reddy also serves as the President of EM Software & Systems (USA) Inc. At EMSS (USA), he is leading the marketing and support of commercial 3D electromagnetic software, FEKO in the US, Canada, Mexico and Central America.

Dr. Reddy is a Senior Member of the IEEE. He is also a member of Applied Computational Electromagetic Society (ACES) and serves as a member of Board of Directors. He has published more than 60 referred journal articles and conference papers.