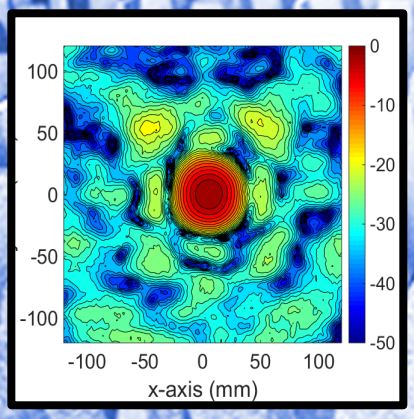
Applied Computational Electromagnetics Society

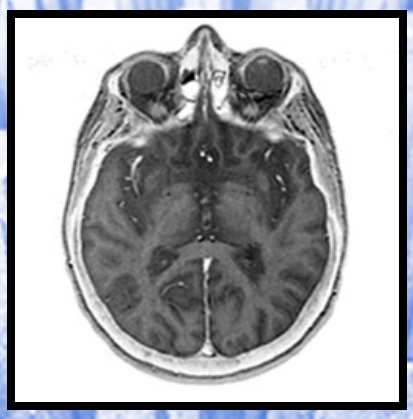
Express Journal

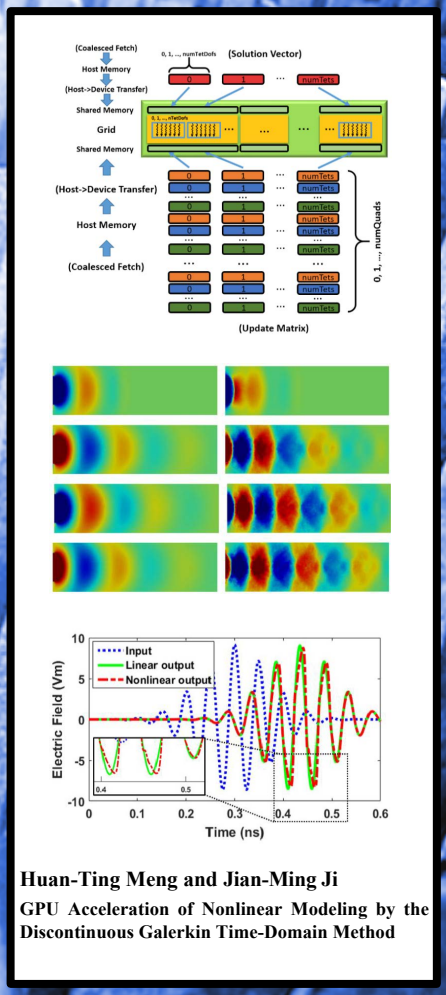
Special Issue On:
GPU Computing in Electromagnetics

Guest Editors: Amedeo Capozzoli, Yahya Rahmat-Samii, and Ozlem Kilic

April 2016 Vol. 1 No. 4







Huan-Ting Meng and Jian-Ming Ji

GPU Acceleration of Nonlinear Modeling by the Discontinuous Galerkin Time-Domain Method

APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY EXPRESS JOURNAL

http://aces-society.org

GENERAL INFORMATION

PURPOSE AND SCOPE: The Applied Computational Electromagnetics Society ***(ACES) Express*** Journal hereinafter known as the ***ACES Express Journal*** is devoted to the timely and rapid exchange of information in computational electromagnetics, to the advancement of the state-of-the art, and the promotion of related technical activities. The primary objective of the information exchange is to inform the scientific community in a short amount of time on the developments of recent computational electromagnetics tools and their use in electrical engineering, physics, or related areas. The technical activities promoted by this publication include code validation, performance analysis, and input/output standardization; code or technique optimization and error minimization; innovations in solution technique or in data input/output; identification of new applications for electromagnetics modeling codes and techniques; integration of computational electromagnetics techniques with new computer architectures; and correlation of computational parameters with physical mechanisms.

SUBMISSIONS: The ***ACES Express Journal*** welcomes original, previously unpublished papers, relating to applied computational electromagnetics. Typical papers will represent the computational electromagnetics aspects of research in electrical engineering, physics, or related disciplines as well as research in the field of applied computational electromagnetics.

Manuscripts are to be submitted through the upload system of ***ACES*** web site http://aces-society.org Please see "Information for Authors" on inside of back cover and at ***ACES*** web site. For additional information contact the Editor-in-Chief:

Dr. Ozlem Kilic
Department of Electrical Engineering and Computer Science
The Catholic University of America
Washington, DC 20064
Email: kilic@cua.edu

SUBSCRIPTIONS: All members of the Applied Computational Electromagnetics Society are entitled to access and download the ***ACES Express Journal*** of any published journal article available at http://aces-society.org. ***ACES Express Journal*** is an online journal and printed copies are not available. Subscription to ACES is through the web site.

LIABILITY. Neither ***ACES***, nor the ***ACES Express Journal*** editors, are responsible for any consequence of misinformation or claims, express or implied, in any published material in an ***ACES Express Journal*** issue. This also applies to advertising, for which only camera-ready copies are accepted. Authors are responsible for all information contained in their papers. If any material submitted for publication includes material which has already been published elsewhere, it is the author's responsibility to obtain written permission to reproduce such material.

THE APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY EXPRESS JOURNAL

Vol. 1 No. 4 April 2016

TABLE OF CONTENTS

The Success of GPU Computing in Applied Electromagnetics

A. Capozzoli [1], O. Kilic [2], C. Curcio [1], and A. Liseno [1]

[1] Università di Napoli Federico II
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
via Claudio 21, I 80125 Napoli, Italy
a.capozzoli@unina.it

[2] The Catholic University of America
Department of Electrical Engineering and Computer Science, Washington, DC
kilic@cua.edu

Abstract ─ In the field of electromagnetic modeling, whether it is the complex designs for engineered materials or devices and components integrated within their natural environments, there is a big drive for highly efficient numerical techniques to model the performance of complex structures. This often cannot be achieved by conventional computer systems, but rather through using the so-called high performance computing (HPC) systems that utilize hardware acceleration. We review recent General Purpose Graphics Processing Units (GPGPU) computing strategies introduced in four fields of computational electromagnetics: Finite-Difference Time-Domain (FDTD), Finite Elements Method (FEM), Method of Moments (MoM) and ElectroMagnetic Ray Tracing (EMRT).

Index Terms ─ CUDA, ElectroMagnetic Ray Tracing (EMRT), Finite-Difference Time-Domain (FDTD), Finite Elements Method (FEM), Graphics Processing Units (GPUs), Method of Moments (MoM), OpenCL, parallel programming.

I. INTRODUCTION

Electromagnetic simulators are essential tools in the analysis and the design of large and complex systems. The last two decades have witnessed dramatic improvements in both algorithms for computational electromagnetics and computing hardware. For the latter point, the use of General Purpose computing on Graphics Processing Units (GPGPU) has become increasingly prevalent. Due to their many computational cores, GPGPUs are indeed suitable for solving problems with a high degree of parallelism.

Successful applications of GPGPU computation require appropriate code implementations and optimizations, depending on whether the problem is memory bound (most of the time spent in memory transactions) or compute bound (most of the time spent in using the GPU) [1]. Throughout the literature, there are several success stories in GPGPU computing as applied to computational electromagnetics. The purpose of this review paper is to sketch the latest GPU computing strategies adopted in four fields of particular interest; namely Finite-Difference Time Domain (FDTD), Finite Elements Method (FEM), Method of Moments (MoM) and ElectroMagnetic Ray Tracing (EMRT). For each of the mentioned representative fields, we will point out the critical aspects, which enable achieving high performance in computations. Also, we will provide relevant references, which will help the interested reader for further details. Finally, nowadays, desktop computers can easily fit four GPUs although, if more computational resources are required, multiple GPUs can be clustered together or heterogeneous systems can be used for large scale simulations. How multi-GPU and heterogeneous systems help increasing the computational performance for the mentioned applications will also be discussed.

II. FDTD

FDTD is one of the most widely used numerical methods for electromagnetic simulations. From the computational point of view, it essentially amounts at stencil calculations. Therefore, the main issue of FDTD is the very low arithmetic intensity, which means that the attainable performance in terms of Floating Point Operations per Second (FLOPS) is limited by the memory bandwidth [2].

Typical strategies like optimizing the arithmetic instructions or hiding the latency of the global memory access by maximizing the multiprocessor occupancy are not effective. For this reason, essentially the optimization approaches below have been applied to GPU-based FDTD implementations for different GPU architectures:

1. Exploit shared memory;
2. Achieve global memory coalesced accesses;

Submitted On: July 11, 2016
Accepted On: July 13, 2016

3. Use the texture cache;
4. Use built-in arrays;
5. Properly arrange the computation in the 3rd dimension.

Concerning point #1, the calculation of field components depends, at each time step, on the value of the same component at the previous step, and on other field components at neighboring cells. Accordingly, it was proposed in [3] to use shared memory to cache all the needed field components, including those corresponding to adjacent computational tiles. In this way, it is possible to significantly reduce data read redundancy. The use of shared memory also enables to limit uncoalesced acceses, as for point #2, see [4].

Regarding point #3, texture memory buffers data in a suited cache, optimized for two-dimensional spatial locality. This leads to performance gains when threads read locations that are spatially close, as in FDTD [4]. However, this benefit appears to be less relevant for latest architectures due to their newly available caching mechanisms.

Concerning point #4, built-in arrays have two, three or four components accessible which allow to best exploit global memory bandwidth. They are used to minimize the number of access operations by maximizing the number of bytes simultaneously transferred [4].

Finally, a very important point in 3D FDTD is the organization of the computation in the third dimension. An efficient solution has been proposed in [3] and a discussion of this topic, in particular, on different solutions proposed in the literature has been recently provided in [5]. An approach to reduce thread divergence when applying Convolutional Perfectly Matched Layer (CPML) boundary conditions has been also proposed in [6].

Compared to a typical implementation on multicore CPUs, an optimized parallelization on GPUs reaches a speedup of the order of ten times. By properly overlapping computation and communication, high parallelization efficiencies (~75%) can be achieved in these cases [7].

III. FEM

The Finite Element Method (FEM) is one of the most advanced and powerful methods for solving Maxwell's equations. Although often used in computational electromagnetics, GPU research on FEM has not been yet as popular as for other numerical methods. Solving Maxwell's equations using FEM essentially consists of three phases [8]:

(i) *Local Assembly*: For each element *e* in the domain, an $N \times N$ matrix, $\underline{\underline{M}}_e$ (*local matrices*), and an N-length vector, $\underline{b}_e$ (*local vectors*), are computed, where N is the number of nodes per element. The computation of $\underline{\underline{M}}_e$ and $\underline{b}_e$ usually involves the evaluation of integrals over the element using Gaussian quadrature. Since meshes are typically unstructured, gathering the data associated with each element forces highly irregular memory accesses.

(ii) *Global Assembly*: The matrices $\underline{\underline{M}}_e$ and the vectors $\underline{b}_e$ are used to form a global matrix $\underline{\underline{M}}$ and global vector $\underline{b}$ by assembling the contributions of the elements together. Typically, $\underline{\underline{M}}$ is very sparse, although its sparsity depends on the connectivity of the mesh. The Compressed Sparse Row (CSR) format is often used to reduce the storage requirement of the matrix and to eliminate redundant computations.

(iii) *Solution of the Relevant Linear System*: The sparse system $\underline{\underline{M}}\,\underline{x} = \underline{b}$ is solved for $\underline{x}$.

There are different possible ways of parallelizing the first two steps. Unfortunately, until now, there is no definite answer on which is the most promising approach. Different techniques are discussed in [8] that are fairly general and relevant to many types of computations on unstructured meshes. A range of possible implementations is presented and recommendations to potential implementers are given. In particular, three possibilities have been considered depending on what each thread is assigned to:

1. Assembly by non-zero elements (each thread is assigned to a different non-zero global matrix element);
2. Assembly by rows (each thread is assigned to a different row of the global matrix);
3. Assembly by elements (each thread is assigned to a different finite element).

Some results have been published for electromagnetic problems in [9] using OpenCL and in [10] using CUDA. A speedup of 19 has been observed for the former case against a multi-core CPU implementation, while a speedup between 87 (matrix assembly) and 51 (solution of the linear system) has been reported for the latter case.

IV. MOM

Method of Moments is another powerful tool used widely in computational electromagnetics. Radiation and scattering problems can be solved numerically using various formulations of the MoM (e.g., EFIE, CFIE, etc.), which is a well-established full-wave analysis based on meshing the geometry into coalescent triangles. The technique employs the expansion of the surface currents of the mesh into a set of basis functions, such as the well-known Rao-Wilton-Glisson (RWG), [11]. The series expansion results in a linear system as expressed as $[V]=[Z].[I]$, where V represents the source function, I is the unknown current, and Z is the impedance matrix. The size of the linear system; i.e., NxN, depends on the number of non-boundary edges in the triangular mesh, N. In the conventional MoM approach, first the

impedance matrix is computed. Then it is inverted, and the unknown currents are calculated. The source vector is computed based on the geometry and the excitation fields at each triangle, [11].

The direct solution of MoM by a matrix inversion presents a big challenge as the object size increases. This is due to the computational complexity, $O(N^3)$, and storage requirements, $O(N^2)$ of MoM. While one way to address the complexity problem is the use of iterative solvers, MoM remains computationally expensive for electrically large objects. The Fast Multipole Method (FMM), which was first introduced by Rokhlin [12] as an augmentation to MoM, reduces the computational complexity for such problems to $O(N_{it}N^2)$ without a significant loss of accuracy. In FMM, the N edges in the mesh are classified into M localized groups, such that each group supports approximately N/M edges. The groups are then categorized as near and far, based on their spatial proximity, allowing the system matrix to be split into, Z_{near} and Z_{far} components, which describe the near and far interactions among the edges. A few authors have applied FMM for electromagnetic problems using a single GPU for small size problems [13], or a GPU cluster for larger problems [14], [15].

Further enhancements have evolved to handle larger problems, such as FMM-FFT, which applies FFT at the translation and multipole expansion stages of FMM, which reduces the complexity to O(NlogN) for two-dimensional rough surfaces, [16] and to $O(N^{4/3}$ log2/3N) for three-dimensional objects, [17]. Recently, FMM-FFT was implemented on a multi-node GPU cluster to demonstrate significant acceleration in computation time while preserving the scalabilty of FMM, [18]. However, FMM-FFT still suffers from the limitation of the GPU memory to solve for larger problems. Another such attempt to enhance FMM for larger scale problems is by introducing a multi-level tree structure of MLFMA, which reduces the computational complexity of MoM to O(NlogN).

V. RAY TRACING

Geometrical Optics (GO) is appealing for scenes with electrically large objects as it provides approximate solutions to Maxwell's equations. In such cases, GO can benefit from the use of data structures inherited by computer graphics, as the Binary Bounding Volume Hierarchies (BBVH), to properly handle the intersections between rays and scene objects.

Ray tracing for GO involves two main steps: searching for the intersections between rays and geometric primitives (for example, triangles) discretizing the object surfaces, and electromagnetic field transport. The first step can be the most time consuming, and must be properly managed. A simple brute force approach would be unfeasible due to the large number of intersection tests to be issued.

This intersection problem can be faced by introducing objects of simple geometry helping in determining if the ray intersects the generic primitive or not, as well as organizing primitives and objects into proper (usually binary) tree hierarchies to reduce the number of intersection tests. Typically, such objects are Axis Aligned Bounding Boxes (AABB). An AABB encloses a group of geometrical primitives or even other bounding volumes. The leaf nodes contain the primitives while the inner nodes enclose the bounding volume of its child nodes. With such a hierarchy, a tree-search algorithm is used to find the nearest object that is hit by a ray. Generally, two schemes are the most popular to construct the hierarchy, namely, *spatial subdivision* and *object partitioning*.

With spatial subdivision, space is recursively split. Each primitive is placed into all leaf nodes to which it overlaps and straddling primitives are copied in multiple nodes. Subdividing space with axis aligned planes leads to the so called KD-tree [19].

On the other side, a binary object partitioning scheme recursively subdivides the primitive list in two non-empty and disjoint sub-lists. For each sub-list, the minimum bounding volumes containing all the sub-list primitives is computed. The bounding volumes may partially overlap and the accelerating structure associated to object partitioning scheme is called BVH [20]. Unlike KDtree, each primitive is stored only once.

Object partitioning and spatial subdivision can work together resulting in a hybrid scheme known as Split Bounding Volume Hierarchy (SBVH) [20, 21], see also [22]. Recently, the benefits and the drawbacks of the above schemes have been analyzed with reference to their GPU implementations [22]. It has emerged that:

- The most critical drawback of KD-tree is the high number of primitive duplicates and the tree depth.
- Besides leading to high memory consumption (which is a problem by itself in GPU computing), primitive duplicates and tree depth are responsible of a larger (as compared to BVH) number of inner-node traversal steps, leaf visits and ray-primitive intersection tests.
- BVH, unlike KD-tree, poorly adapts to arbitrary scenes with very varying density. SBVH has shown to be a very satisfactory compromise.

With SBVH, it has recently shown how thousands of millions of rays per second can be traced on a Kepler K20c card [23].

VI. CONCLUSION

We have reviewed recent GPGPU computing strategies introduced in five fields of computational electromagnetics: FDTD, FEM, MoM and EMRT. The purpose has been to provide new Researchers in this field with initial guidelines on the dealt with topics. At present, research in GPU accelerated FEM for electromagnetics surprisingly appears to have been

overlooked in the literature.

REFERENCES

[1] P. Micikevicius, "Identifying performance limiters," *GTC Technology Conf*., 2011.

[2] K.-H. Kim, K. H. Kim, and Q.-H. Park, "Performance analysis and optimization of three-dimensional FDTD on GPU using roofline model," *Computer Phys. Commun.*, vol. 182, no. 6, pp. 1201-1207, June 2011.

[3] P. Micikevicius, "3D finite difference computation on GPUs using CUDA," *Proc. of 2nd Workshop on General Purpose Processing on GPUs*, Washington, DC, USA, pp. 79-84, Mar. 8, 2009.

[4] D. De Donno, A. Esposito, L. Tarricone, and L. Catarinucci, "Introduction to GPU computing and CUDA programming: a case study," *IEEE Antennas Prop. Mag.*, vol. 52, no. 3, pp. 116-122, June 2010.

[5] M. Livesey, J. F. Stack Jr., F. Costen, T. Nanri, N. Nakashima, and S. Fujino, "Development of a CUDA implementation of the 3D FDTD method," *IEEE Antennas Prop. Mag.*, vol. 54, no. 5, pp. 186-195, Oct. 2012.

[6] J. I. Toivanen, T. P. Stefanski, N. Kuster, and N. Chavannes, "Comparison of CPML implementations for the GPU-accelerated FDTD solver," *Progr. Electromagn. Res.*, vol. 19, pp. 61-75, 2011.

[7] R. Shams and P. Sadeghi, "On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters," *J. Parallel Distrib. Comput.*, vol. 71, no. 4, pp. 584-593, Apr. 2011.

[8] C. Cecka, A. J. Lew, and E. Darve, "Assembly of finite element methods on graphics processors," *Int. J. Numer. Meth.*, vol. 85, no. 5, pp. 640-669, Feb. 2011.

[9] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Finite element matrix generation on a GPU," *Progr. in Electromagn. Res.*, vol. 128, pp. 249-265, 2012.

[10] Z. Fu, T. J. Lewis, R. M. Kirby, and R. T. Whitaker, "Architecting the finite element method pipeline for the GPU," *J. Comput. Appl. Math.*, vol. 256, pp. 195-211, Feb. 2014.

[11] S. M. Rao, D. R. Wilton, and A. W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. Antennas Prop.*, vol. AP-30, no. 3. pp. 409-418, May 1982.

[12] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Prop. Mag*., vol. 35, no. 3, pp. 7-12, June 1993.

[13] K. Xu, D. Z. Ding, Z. H. Fan, and R. S. Chen, "Multilevel fast multipole algorithm enhanced by GPU parallel technique for electromagnetic scattering problems," *Microw. Opt. Technol. Lett*., vol. 52, pp. 502-507, 2010.

[14] Q. Nguyen, V. Dang, O. Kilic, and E. El-Araby, "Parallelizing fast multipole method for large-scale electromagnetic problems using GPU clusters," *IEEE Antennas Wireless Prop. Lett.*, vol. 12, pp. 868-871, 2013.

[15] V. Dang, Q. Nguyen, and O. Kilic, "Fast multipole method for large-scale electromagnetic scattering problems on GPU cluster and FPGA accelerated platforms," *Applied Comp. Electromag. Soc. Journal*, Special Issue, vol. 28, no. 12, pp. 1187-1198, 2013.

[16] R. L. Wagner, J. Song, and W. C. Chew, "Monte Carlo simulation of electromagnetic scattering from two-dimensional random rough surfaces," *IEEE Trans. Antennas Prop.*, vol. 45, no. 2, pp. 235-245, 1997.

[17] C. Waltz, K. Sertel, M. A Carr, B. C. Usner, and J. L. Volakis, "Massively parallel fast multipole method solutions of large electromagnetic scattering problems," *IEEE Trans. Antennas Prop.*, vol. AP-55, no. 6, pp. 1810-1816, 2007.

[18] V. Dang, Q. Nguyen, and O. Kilic, "GPU cluster implementation of FMM-FFT for large-scale electromagnetic problems," *IEEE Antennas Wireless Prop. Lett.*, vol. 13, pp. 1259-1262, 2014.

[19] Y. Tao, H. Lin, and H. Bao, "GPU-based shooting and bouncing ray method for fast RCS prediction," *IEEE Trans. Antennas Prop.*, vol. 58, no. 2, pp. 494-502, Feb. 2010.

[20] T. Aila and S. Laine, "Understanding the efficiency of ray traversal on GPUs," *Proc. of the Conf. on High Performance Graphics*, Saarbrucken, Germany, pp. 145-150, June 25-27, 2009.

[21] I. Wald and V. Havran, "On building fast KD-Trees for ray tracing, and on doing that in O(NlogN)," *Proc. of the IEEE Symposium on Interactive Ray Tracing*, Salt Lake City, UT, pp. 61-69, Sept. 18-20, 2006.

[22] A. Breglia, A. Capozzoli, C. Curcio, and A. Liseno, "GPU-based shooting and bouncing ray method for fast RCS prediction," *IEEE Antennas Prop. Mag.*, vol. 57, no. 5, pp. 159-176, Oct. 2015.

[23] A. Breglia, A. Capozzoli, C. Curcio, and A. Liseno, "Why does SBVH outperform KD-tree on parallel platforms?," *Proc. of the IEEE/ACES Int. Conf. on Wireless Inf. Tech. and Syst. and Appl. Comput. Electromagn.*, Honolulu, HI, pp. 1-2, Mar. 13-18, 2016.

Benefits and Challenges of GPU Accelerated Electromagnetic Solvers from a Commercial Point of View

Ulrich Jakobus

Altair Development S.A. (Pty) Ltd.
Stellenbosch, 7600, South Africa
jakobus@altair.com

Abstract ─ This paper discusses the benefits but also challenges of GPU accelerated electromagnetic solvers from a commercial point of view, namely using FEKO as example. Specifically, the effects of some of the complex interdependencies between different components are presented. It is shown that despite the advances made in the field of GPGPU computing, and impressive speedups for parts of a program or simplified problems, there are a number of factors to consider before these techniques can be applied to a commercial product that is expected to be robust and, most importantly, to always give trustworthy results for a wide variety of problems.

Index Terms ─ Commercial Solvers, CUDA, FDTD, FEKO, FEM, GPGPU, GPU Acceleration, MoM, RL-GO, SBR.

I. INTRODUCTION

In the field of computational electromagnetics (CEM), a wide range of numerical techniques can be used to simulate a variety electromagnetic radiation and scattering problems. One of the primary reasons that such a wide variety of methods exists, is that no single method performs best for all problem types [1]. Thus, one of the first challenges in solving an electromagnetic problem is to select the method that is best or at least reasonably suited to the problem of interest.

Even with the optimal method selected, there is still the matter of the available computational resources to consider. It may then be that the desired solution takes hours, days, or even weeks to compute. One of the ways in which an attempt has been made to increase the computational power at disposal – thereby decreasing the time required for a solution – has been to make use of graphics processing units (GPUs) to perform general purpose computational tasks, and not just the graphics-related tasks for which they were originally designed for. This practice, called general purpose GPU (GPGPU) computing, has seen a remarkable increase of late, both in terms of hardware capability, as well as the ease with which these devices can be programmed [2].

The most common way of programming such devices is using the Compute Unified Device Architecture (CUDA) by NVIDIA. This couples a genuinely programmable hardware architecture with programming tools that can be used by any developer with a knowledge of C/C++. Previously, GPGPU programming involved convincing a GPU to do what one wanted by rewriting computational routines as graphics programs. Since its inception, CUDA's hardware/software combination has evolved to such an extent that the latest generation of devices can be found in the fastest supercomputers in the world, with a much more powerful set of software features available as well.

There has been considerable development and a large number of papers were published on the GPU acceleration of CEM methods, for example the Method of Moments (MoM) [3] and [4], the Finite Element Method (FEM) [5] and [6], and the method of Shooting and Bouncing Rays (SBR) [7] and [8]. More general advances such as in GPU based dense linear algebra methods can be found, e.g., in [9]. The focus of this paper is not to add to this (we have done so earlier, e.g., in [10] or [11]), but instead to present an alternate perspective on these advances. That is to say the use of GPU technology as well as the challenges related to it are considered from the point of view of a commercial CEM software. To this end, the software package FEKO [12] is taken as an example. The motivation for this is that quite often such advances are considered from a purely academic standpoint, and this leads to a number of short-comings and challenges being overlooked.

Section II gives a short introduction on the FEKO solution kernel and the various CEM methods that are supported by it. This serves as background for a discussion on the difficulties associated with the GPU acceleration of a commercial CEM software package such as FEKO in Section III, and a short discussion of GPU accelerated solvers that exist in FEKO or are under development in Section IV. The paper is concluded in Section V, where a discussion on future paths to facilitate further GPU acceleration is included.

Submitted On: August 28, 2015
Accepted On: January 24, 2016

II. THE FEKO SOLUTION KERNEL

As already mentioned, a number of CEM methods exist which have their own strengths and weaknesses, and which can solve various problems of interest with varying degrees of success. It is thus important that a commercial CEM code such as FEKO implements a number of these methods to allow it to be competitive for a large selection of target application areas.

Figure 1 shows the various solution techniques available in FEKO for the solution of RF/microwave problems. Two factors influencing the choice of solution method – the electrical size of the problem being considered and the complexity of the materials being simulated – are indicated on the axes. The possibility of hybridizing various methods exists, and this allows for the solution of more complex problems by selecting the best solution method for different regions of the same problem with full bi-directional coupling between them.

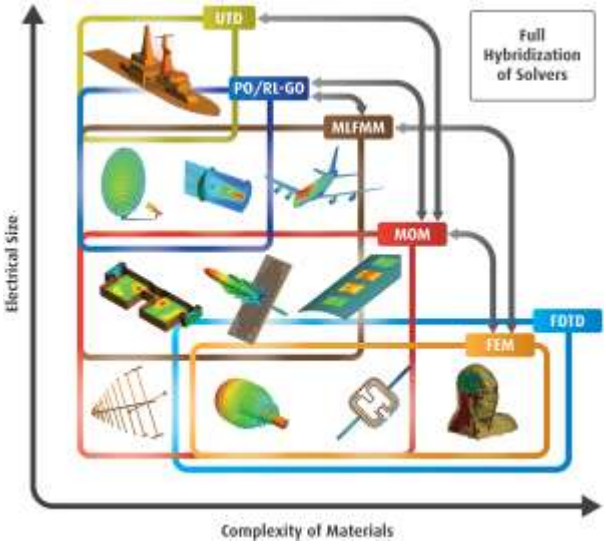


Fig. 1. A diagram depicting the various computational methods in FEKO. The hybridization that exists between some of the methods is also shown by green arrows.

III. CHALLENGES IN GPU ACCELERATION

In any software development, it is required that the available resources be allocated to maximize the delivered value in the software project. How value is determined is specific to each project, and may also differ greatly between the academic and commercial environments. In the commercial environment, for example, the number of customers with capable hardware demanding or being able to use GPU acceleration directly influences the relative value of GPU accelerated extensions when compared to other feature extensions. Academic development may, on the other hand, place a high importance on novelty for use in academic publications.

A. Versatility, reliability, and reproducibility

Many academic publications on the topic of accelerated CEM codes consider a small number of examples to illustrate the applicability or performance improvements of a specific method. These examples are often simple or canonical problems, which may play to the strengths of the method being considered, and also may not exceed the resources – such as available memory – of the GPU being used for acceleration.

In the commercial setting, there is no such control over which examples are being considered, and customers expect accurate results for a wide variety of problems. This not only imposes heavy resource requirements for additional validation and verification of the accelerated methods, but also in the detection of possible problem cases at run-time (such as running out of GPU memory and then switching automatically to block based algorithms or switching the computations on the fly back to the CPU), and handling these in a well-rounded and user friendly way.

B. Variety of CEM methods

The various computational methods included in the FEKO solution kernel and discussed in Section II have their own strengths and weaknesses when it comes to the solution of CEM problems. In addition, each of these methods present its own challenges in parallelization in general (MPI, OpenMP, etc.), and in GPU computing specifically.

Take the Methods of Moment (MoM) and the Finite Element Method (FEM) as examples. These are both matrix-based methods which require the construction, and (for driven problems) the subsequent solution of a linear system of equations. It is also possible to formulate certain classes of problems in each method as generalized eigenvalue problems.

At this point it may seem as if these two methods would be amenable to similar approaches when considering them for GPU acceleration. The situation is, however, that the linear system which results as part of a MoM computation is dense, whereas that associated with the FEM is a sparse system. Although GPU tools exit for the solution of both types of systems, the difference in performance of dense and sparse computation on a GPU means that the realized speedup will differ significantly. Furthermore, the effect of the other phases in the solution process (e.g., matrix fill) must also be taken into consideration and will be discussed in Section IV.

C. Software and design decisions

Another important factor regarding the adoption of GPU acceleration in an existing commercial CEM package are design and development decisions such as the language of implementation and low-level program

flow, which if – if not selected carefully – may not map well to massively parallel architectures such as GPUs.

CUDA was already mentioned as programming language to support NVIDIA cards. In the OpenMP 4.0 standard, for example, provision has been made for the use of accelerators. OpenMP is a directive-based, open standard which provides a portable means to parallelize code over a number of threads. The inclusion of the concept of accelerators and the associated operations, means that the importance of such technologies has been recognized. Furthermore, since the directives are platform agnostic, acceleration would in theory not be limited to a particular set of devices – such as NVIDIA GPUs when using CUDA – but the same code could be used to run on multi-core CPUs, GPUs by other vendors, and other accelerator technologies such as Intel's Xeon Phi coprocessors. There is also OpenCL, kind of being in the middle between CUDA and OpenMP. In FEKO, all three techniques (OpenCL, OpenMP, and CUDA) are being explored and partially used, but all the following GPU discussions refer to CUDA specifically.

Considering that many of the GPGPU programming tools are centered on C/C++ implementations, the options for the acceleration of for instance FORTRAN based routines generally involve rewriting large portions of code in C/C++, or switching to FORTRAN compilers that do support GPU computing. Any rewriting introduces the risk of introducing new bugs, increasing the need for proper tuning, testing and software verification.

In terms of switching compilers, there are also a number of factors to consider. One of the biggest problems is the loss of productivity – possibly for a whole development team – due to changes required in build processes and utilities, the introduction of unforeseen bugs caused by incompatible compiler options, and bugs in the compilers themselves.

IV. GPU ACCELERATION IN FEKO

A. The Method of Moments

As discussed in Section III, the MoM requires the assembly and solution of a dense linear system with other steps followed like near or far field calculations. The run-time for the assembly of the matrix is quadratic in terms of the number of unknowns, whereas that of the solution of the linear system is cubic. The post-processing is typically linear in terms of the number of unknowns and linear in terms of the number of far field directions/near field observation points etc. It follows that as the problem size gets larger, the matrix solution phase will dominate the overall run-time.

The matrix solution phase can be isolated and accelerated using libraries such as MAGMA [9] or cuSOLVER (available as part of CUDA since version 7.0). Unfortunately, even though it can be accelerated by up to an order of magnitude, the total simulation acceleration is significantly less, with the matrix assembly phase now dominating the run-time. Even though considerable speedups can be attained for this matrix fill phase in simplified MoM code [3], a considerable amount of development resources need to be invested for a FEKO implementation due to the complex nature of the code (many different basis functions, higher order on curvilinear meshes, Sommerfeld integrals for planar Green's functions etc.).

B. The Finite Element Method

Another matrix-based method implemented in FEKO is the FEM. In contrast to the MoM, the matrices are sparse, but many of the same challenges present themselves when the GPU acceleration of the method is considered.

Here, the phases of the solution process which contribute most significantly to the total simulation time are the construction of the relevant preconditioner and the subsequent solution of the sparse linear system. FEKO uses by default iterative solvers for a single right hand side which – with the right preconditioners – provide according to our experience faster solution times than direct sparse solvers and in particular use less memory.

For the solution of FEM linear system, a simple iterative solver can be expected to show a 2-5x performance improvement when running on a GPU, but for most problem sizes where the amount of GPU memory is not a limitation, this translates into a simulation speedup of only 50% as the other phases start dominating.

Further acceleration is hampered by the sheer number of preconditioning options available in a software such as FEKO. In addition, differences in matrix representation and the lack of complex value support in available third-party libraries make the use of a standalone approach – as was done with the MoM matrix solution – problematic.

C. Ray launching Geometrical Optics

Along Uniform Theory of Diffraction (UTD) and Physical Optics (PO), the Ray Launching Geometrical Optics (RL-GO) solver – which is sometimes referred to as Shooting and Bouncing Rays (SBR) – is ideal for the analysis of electrically large and complex objects. It is inherently parallel and is well suited to GPU acceleration. As an initial proof of concept, we were able to accelerate the calculation of the intersections of rays with geometry in FEKO by at least an order of magnitude when using CUDA.

However, this was handwritten CUDA code. It is not possible to simply run the RL-GO code through a GPU aware compiler and obtain an accelerated implementation with similar performance. Furthermore, the complexity and recursive nature of the code means

that GPU specific limits such as smaller stack size must be addressed as well.

D. Finite Difference Time Domain Method

In much the same way that the RL-GO solver is algorithmically well suited to GPU acceleration, the Finite Difference Time Domain (FDTD) method lends itself well to such parallelization. Much of this stems from the fact that the same simple update equations are applied to each voxel in each time step with (almost) no communication required between adjacent updates. As is indicated in Fig. 1, one advantage of the acceleration of the FDTD over the RL-GO solver in FEKO is that there is as yet no hybridization of FDTD with other methods and thus, less complexity to be considered.

The FDTD solver implemented in FEKO makes used of GPU acceleration to provide roughly an order of magnitude speedup for certain problems. One disadvantage of such a speedup is that from a user's perspective, the relative performance of post-processing phases such as the calculation of far fields is significantly lower.

For both CPU and GPU based FDTD solvers, the measured performance is greatly affected by the problem setup, which includes factors such as user-requested near fields or the far fields already mentioned. If these are in the frequency domain, for example, then additional costly computations are required during every simulation time step.

V. CONCLUSION

In this paper, a discussion on the challenges associated with the GPU acceleration of the commercial CEM software package FEKO was presented. This showed that although a method may be promising theoretically, its application in commercial software generally requires the allocation of significant development resources, with at this stage not always the necessary demand from the market.

As examples, the acceleration of the MoM, FEM, and RL-GO were considered, and although certain phases of the computational process can be accelerated significantly, the total simulation speedup is limited. The further acceleration of these methods is hampered by the complexity of the numerical algorithms, e.g., through hybridization. As illustrated, for FDTD, the situation is different.

REFERENCES

[1] D. B. Davidson, *Computational Electromagnetics for RF and Microwave Engineers*, 2nd ed., Cambridge: Cambridge University Press, 2011.

[2] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors – A Hands-on Approach*, Burlington: Morgan Kaufmann, 2010.

[3] E. Lezar and D. B. Davidson, "GPU-accelerated methods of moments by example: monostatic scattering," *IEEE Antennas and Propagation Magazine*, vol. 52, no. 6, pp. 120-135, Dec. 2010.

[4] M. J. Inman, A. Z. Elsherbeni, and C. J. Reddy, "CUDA based GPU solvers for method of moment simulations," *Annual Review of Progress in Applied Computational Electromagnetics*, Tampere, Finland, Apr. 2010.

[5] E. Lezar and D. B. Davidson, "GPU-based Arnoldi factorization for accelerating finite element eigenanalysis," *International Conference on Electromagnetic in Advanced Applications (ICEAA)*, Torino, Italy, Sept. 2009.

[6] A. Dziekonski, A. Lamecki, and M. Mrozowski, "On fast iterative solvers with GPU acceleration for finite elements in electromagnetics," *10th International Workshop on Finite Elements for Microwave Engineering*, Mill Falls, NH, USA, Oct. 2010.

[7] K. E. Spagnoli, *An Electromagnetic Scattering Solver Utilizing Shooting and Bouncing Rays Implemented on Modern Graphics Cards*, ProQuest, 2008.

[8] Y. Tao, H. Lin, and H. Bao, "GPU-based shooting and bouncing ray method for fast RCS prediction," *IEEE Trans. Antennas Propagat.*, vol. 58, no. 2, pp. 494-502, 2010.

[9] ICL, University Tennessee, Knoxville, "MAGMA: Matrix Algebra on GPU and Multicore Architectures," 2015. [Online]. Available: http://icl.cs.utk.edu/magma/index.html

[10] E. Lezar and U. Jakobus, "GPU accelerated electromagnetic simulations with FEKO," *International Supercomputing Conference*, Hamburg, June 2012.

[11] E. Lezar, U. Jakobus, and S. Kodiyalam, "GPU related advances in the FEKO electromagnetic solution kernel," *International Conference on Electromagnetics in Advanced Applications (ICEAA)*, Torino, Italy, Sept. 2013.

[12] Altair Development S.A. (Pty) Ltd, "FEKO – Field Computations Involving Bodies of Arbitrary Shape," 2016. [Online]. Available: www.altairhyperworks.com/feko

GPU Acceleration of Nonlinear Modeling by the Discontinuous Galerkin Time-Domain Method

Huan-Ting Meng and Jian-Ming Jin

Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
meng2@illinois.edu, j-jin1@illinois.edu

Abstract ─ A discontinuous Galerkin time-domain (DGTD) algorithm is formulated and implemented to model the third-order instantaneous nonlinear effect on electromagnetic fields due the field-dependent medium permittivity. The nonlinear DGTD computation is accelerated using graphics processing units (GPUs). Two nonlinear examples are presented to show the different Kerr effects observed through the third-order nonlinearity. With the acceleration using MPI + GPU under a large cluster environment, the solution times for nonlinear simulations are significantly reduced.

Index Terms ─ Computational electromagnetics, DGTD, GPU acceleration, Kerr effect, nonlinear electromagnetics, third-order nonlinearity.

I. INTRODUCTION

Nonlinear phenomena in electromagnetics generally involve changes in the material properties due to the presence of electromagnetic fields. The changes in the material properties in turn modify the state of the original electromagnetic fields in the medium. Since the material properties and the contained fields interact with each other constantly, it is most natural to describe and model these interactions in the time domain, where at each time instant the changes in the fields induce nonlinear modifications on both the material properties and the fields themselves.

The nonlinear Kerr effect [1] is one of the most studied and exploited optical effects. It describes the third-order interaction between the electric field and the permittivity of the material, which produces a variety of nonlinear phenomena [1], [2], such as third-harmonic generation (THG), self-phase modulation (SPM), self-focusing, and frequency mixing. Much investigation has been carried out for the simulation of the nonlinear optical effects using the finite-difference time-domain (FDTD) algorithms [3], due to their straightforward implementation.

This work is focused on the modeling of the third-order Kerr instantaneous nonlinearity using the discontinuous Galerkin time-domain (DGTD) algorithm. The nonlinear DGTD algorithm possesses many advantages of the linear DGTD algorithms over nonlinear FDTD algorithms, including the flexibility in complex geometry modeling, reduced phase shifts, and the ease to achieve higher order accuracy and convergence. To speed up the computation, the MPI + GPU framework developed in [4] is adapted to accelerate the nonlinear DGTD algorithm.

II. FORMULATION

For a general third-order nonlinear medium, the relative permittivity can be written as:

$$\varepsilon_r = \varepsilon_r(E) = \varepsilon_{r,\mathrm{L}} + \varepsilon_{r,\mathrm{NL}} = \varepsilon_{r,\mathrm{L}} + \chi^{(3)}E^2, \tag{1}$$

where $\varepsilon_{r,\mathrm{L}}$ and $\varepsilon_{r,\mathrm{NL}}$ are the linear and nonlinear parts of the relative permittivity, respectively, $\chi^{(3)}$ is the third-order nonlinear polarization coefficient, and E is the magnitude of the time-varying electric field. Here we focus on the derivation of the DGTD algorithm to model a nonlinear, lossless, and non-dispersive medium to update the electric field since the updating equation for the magnetic field has no nonlinear components and thus is identical to that in a linear medium. Testing Ampere's law using the Galerkin method, substituting in the expansion of the fields, and applying the central flux, the equation after taking the time derivative on $\vec{D}$ for element e becomes:

$$\begin{aligned}[S_e]\{h\} + \iiint_{V_e}\left[\varepsilon_0\frac{\partial\varepsilon_r}{\partial t}\vec{N}_i^e\cdot\vec{N}_j^e\right]dV\{e\} \\ + \iiint_{V_e}\left[\varepsilon_0\varepsilon_r\vec{N}_i^e\cdot\vec{N}_j^e\right]dV\frac{\partial\{e\}}{\partial t} = [F_{eh}]\{h^+ - h\},\end{aligned} \tag{2}$$

where

$$S_e(i,j) = \iiint_{V_e}\frac{1}{\mu_\infty}(\nabla\times\vec{N}_i^e)\cdot(\nabla\times\vec{N}_j^e)\,dV, \tag{3}$$

and $\{e\}$ and $\{h\}$ are the electric and magnetic field solution vectors and $\vec{N}_i^e$ and $\vec{N}_j^e$ are vector basis functions. The terms associated with the boundary conditions are omitted for simplicity. Since the time-varying permittivity is embedded in the mass matrix of

Submitted On: August 29, 2015
Accepted On: March 30, 2016

the DGTD algorithm, the volume integration pertaining to the electric field is now split into two terms by the product rule, where for the nonlinear medium, both the relative permittivity and the electric field are functions of time. Discretizing Equation (2) in the time domain using central difference gives:

$$\iiint_{V_e}\left[\varepsilon_0\frac{\varepsilon_r^{n+1}-\varepsilon_r^{n}}{\Delta t}\vec{N}_i^e\cdot\vec{N}_j^e\right]dV\left(\frac{\{e\}^{n+1}+\{e\}^{n}}{2}\right)$$
$$+\iiint_{V_e}\left[\varepsilon_0\frac{\varepsilon_r^{n+1}+\varepsilon_r^{n}}{2}\vec{N}_i^e\cdot\vec{N}_j^e\right]dV\left(\frac{\{e\}^{n+1}-\{e\}^{n}}{\Delta t}\right) \quad (4)$$
$$=\{b\}^{n+1/2},$$

where ε_r^{n+1} is the field-dependent nonlinear permittivity at the future time step, ε_r^{n} is the converged permittivity at the current time step, and

$$\{b\}^{n+1/2}=[F_{eh}](\{h^+\}^{n+1/2}-\{h\}^{n+1/2})-[S_e]\{h\}^{n+1/2}. \quad (5)$$

After rearranging the terms, Equation (4) can be cast into a field-marching form as:

$$[M_e]^{n+1}\{e\}^{n+1}-[M_e]^{n}\{e\}^{n}=\{b\}^{n+1/2}, \quad (6)$$

where

$$[M_e]^{n+1}=\frac{\varepsilon_0}{\Delta t}\iiint_{V_e}\left[\varepsilon_r^{n+1}\vec{N}_i^e\cdot\vec{N}_j^e\right]dV, \quad (7)$$

and

$$[M_e]^{n}=\frac{\varepsilon_0}{\Delta t}\iiint_{V_e}\left[\varepsilon_r^{n}\vec{N}_i^e\cdot\vec{N}_j^e\right]dV. \quad (8)$$

Due to the variation of the field magnitude at each time step, $\varepsilon_r(E)$ of each element changes with time, and therefore the mass matrix $[M_e]^{n+1}$ has to be reassembled at every time step. Note that, we have recovered the original expression for $[M_e]$ as in the linear DGTD algorithm, albeit with a field- and time-dependent permittivity. The dependency of $\{e\}^{n+1}$ in $[M_e]^{n+1}$ renders Equation (6) a nonlinear equation.

At each time marching step *n*, the fixed-point method is employed to solve Equation (6), where $\{b\}^{n+1/2}$ is computed with the initial guess $\{e\}_0^{n+1}=\{e\}^{n}$ and $[M_e]_0^{n+1}=[M_e]^{n}$. At the *k*th iteration step, the mass matrix $[M_e]_{k-1}^{n+1}$ is inverted to update the field solution $\{e\}_k^{n+1}$. The updated solution is in turn used to update the mass matrix $[M_e]_k^{n+1}$ using Equation (7). If the norm of the residual $\{r\}_k^n$ of Equation (6) is smaller than a predefined threshold, then the nonlinear iteration is converged, and the equation can be marched to the next time step $n+1$. Otherwise it continues with the $(k+1)$th iteration step.

III. GPU IMPLEMENTATION

Because of the necessity to solve nonlinear equations in each time step, the nonlinear DGTD computation is very time-consuming. This computation can be effectively accelerated by exploiting the power of graphics processing units (GPUs). The GPU implementation for the nonlinear DGTD algorithm is similar to the approach described in [4], employing the same coalesced memory accessing pattern and thread/block allocation. Since the electric field update processes that are not related to $\{e\}^{n+1}$ are similar to the ones found in [6], here we focus on the parallelization of the computation related to $\{e\}^{n+1}$, which includes the assembly of the nonlinear mass matrix $[M_e]^{n+1}$ and the inversion of this mass matrix.

To assemble the nonlinear mass matrix, note that each mass matrix entry is numerically integrated through quadrature, where the contribution from each weighted quadrature point is summed. Due to the presence of nonlinearity, ε_r on each quadrature point changes during each iteration step, while the other constituting terms in equation (7) remain identical. To parallelize the assembly of the mass matrix, the constituting matrices at each quadrature point are pre-calculated and stored, and then summed together at each iteration step by first multiplying with the updated ε_r. The proposed parallelization strategy and the memory access pattern are shown in Fig. 1, with each of the total *numTets* elements parallelized over its *numTetDofs* unknowns using CUDA threads. Each threadblock is assigned with a calculated number of elements to utilize all warps [4]. At each iteration step, the mass matrices are assembled by looping through *numQuads* quadrature points and summing their contribution, which is completely parallelizable.

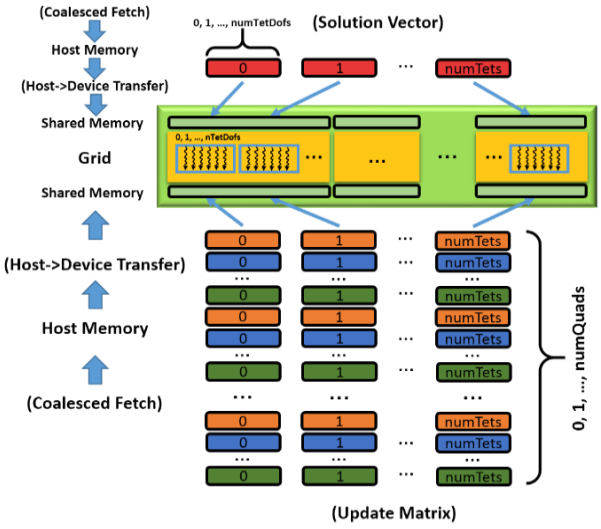


Fig. 1. Parallelization and memory access pattern for the assembly of the nonlinear mass matrices.

To invert the nonlinear mass matrix, we parallelize the standard non-pivoting element-level Gaussian elimination on the GPU. Each *numTetDofs threads* for an element loops over each elemental matrix rows and

reduce them into row echelon form. Although the elimination is only semi-parallelizable, the batch processing of the elimination process for the nonlinear elements somewhat provides a decent speedup. Note that, the mass matrix has a small condition number, and therefore can be easily inverted using the standard Gaussian elimination without partial pivoting. This is beneficial for the GPU acceleration since the partial pivoting process involves many conditional statements and branches, which are undesirable for the parallelization on GPUs.

IV. NUMERICAL EXAMPLES

Two examples are presented here to demonstrate the self-phase modulation, the third-harmonic generation, and the self-focusing effects captured by the extended DGTD algorithm and the GPU speedup. The simulation was carried out on the XSEDE Stampede cluster with NVIDIA Tesla K20 GPUs and Xeon E5-2680 CPU threads.

A. Demonstration of the self-phase modulation and the third-harmonic generation

The first example is a coaxial waveguide with an inner and outer radius of 1 and 2 mm, respectively, and a length of 40 mm. A small section of linear medium is placed near each end for excitation and absorption of the fields, and the rest of the coaxial waveguide is filled with either a linear or nonlinear medium, with a linear permittivity of $\varepsilon_{r,\mathrm{L}} = 1.0$ and a third-order nonlinearity coefficient of $\chi^{(3)} = 4\text{e-}8$. The input signal is a modulated Gaussian pulse with a center frequency of 20 GHz. The number of finite elements is 110,715, and the solution marches at a time step of $\Delta t = 0.075\text{ps}$ for a total of 10,000 time steps for both the linear and nonlinear cases. Mixed first-order basis functions are used for the computation. The time-domain response for the two cases is shown in

Fig. 2. It can be observed that with a linear medium, the shape of the output signal is identical to the input, whereas with a nonlinear medium the output signal steepens and forms shock waves, showing the self-steepening effect [1].

The frequency-domain response for the output signal is shown in

Fig. 3. For the linear case, we have retained the frequency profile of the original input Gaussian pulse centered at 20 GHz. For the nonlinear case, the third-harmonic effect generates harmonics at odd multiples of the original 20 GHz signal at 60 GHz, 100 GHz, 140 GHz, and so on. In addition, the self-phase modulation effect broadens the input bandwidth, where the leading and the trailing edges shift to lower and higher frequencies, respectively [1]. This result is validated using COMSOL. Table 1 gives the average per-step CPU and GPU timing for the simulation. The lower speedup as comparing to [4] is in large due to the uneven nonlinearity encountered by the different elements, which correlates to thread idling in a warp, and the semi-serial nature of the Gaussian elimination process. This thread idleness effectively lowers the number of FLOPS as well as the overall bandwidth.

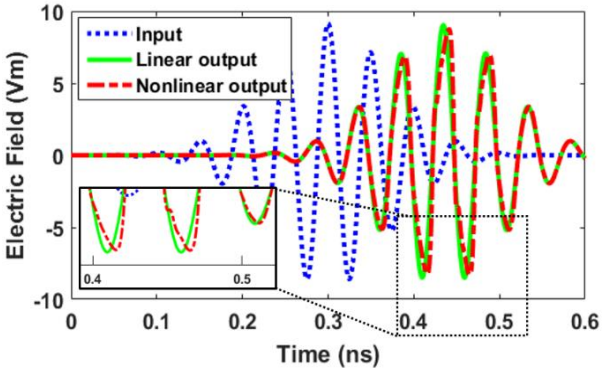


Fig. 2. Time-domain response of the electric field for a coaxial waveguide filled with a section of linear or nonlinear medium.

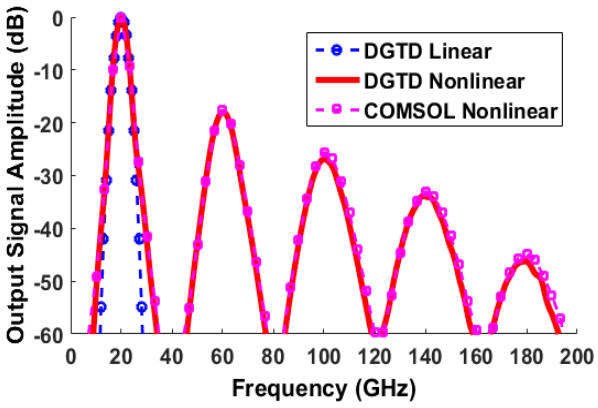


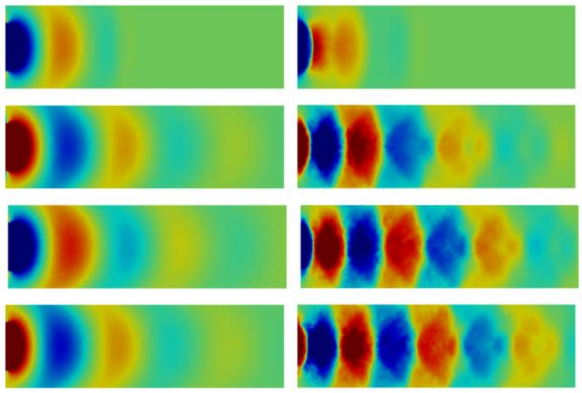
Fig. 3. Frequency-domain response of the output signal for a coaxial waveguide filled with a section of linear or nonlinear medium.

Table 1: Average per-step timing comparison for the simulation of a nonlinear coaxial waveguide

# MPI	1	2	4	8
	CPU Time per Step (ms)			
Marching	1,482.00	741.61	369.71	183.15
Comm.	0	35.51	33.74	38.79
Per-Step	1,482.00	777.12	403.45	221.94
	GPU Time per Step (ms)			
Marching	47.21	23.73	12.00	6.14
Comm.	0	2.94	1.53	4.57
Per-Step	47.21	26.67	13.52	10.71
Speedup	31.39	29.14	29.83	20.72

B. Demonstration of the self-focusing effect

The second example demonstrates the self-focusing effect through beam-shaped field propagation in a $1\text{mm}\times1\text{mm}\times3\text{mm}$ bulk medium. The linear relative permittivity is $\varepsilon_{r,\text{L}}=1.0$ and the third-order nonlinearity coefficient is $\chi^{(3)}=8$. The excitation is a tapered TEM sine wave at 300 GHz, launched through a square aperture with a dimension of a half of the excitation wavelength. The number of finite elements is 664,039, and the solution marches at a time step of $\Delta t=0.01\text{ps}$ for a total of 5,000 time steps, where mixed first-order basis functions are used for the simulation. The field profiles in the bulk medium at various times for both linear and nonlinear cases are shown in Fig. 4. In the nonlinear medium, the specific electric field generates a strong nonlinearity, which results in a maximum instantaneous relative permittivity of $\varepsilon_r=8.27$, or a 727% change to the linear relative permittivity. As can be seen, due to nonlinearity, the field experiences pulse compression which shortens the duration of each pulse. This effect is due to self-phase modulation. As the field propagates along the bulk medium, the wave is naturally diffracted in the linear medium, where the magnitude of the field decreases significantly after a couple of wavelengths. In the nonlinear medium, the intensity of the field modifies the surrounding medium into a self-induced waveguide, which counteracts natural diffraction and preserves the magnitude of the propagating wave for a longer distance in the medium.



(a) (b)

Fig. 4. Time-domain field profile for wave propagation in a: (a) linear and (b) nonlinear medium at 5, 20, 25, and 50ns, respectively.

Table 2 shows the GPU average per-step timing. Since different elements experience different levels of nonlinearity at different times due to the propagation of the field, the CUDA threads for a converged element will idle and wait for the rest of the elements in the same GPU to synchronize before completing the kernel (a single time step). This results in some MPI nodes having to idle and wait for the others to iteratively converge before moving onto the next time step together. This idling time is taking into account in the average communication time, which is significantly longer for the fixed-point method due to the large differences in the number of iterations between different regions at any particular moment. Due to the high nonlinearity of the example, it is impractical to analyze the CPU performance. However, it is expected that higher speedup can be achieved comparing to the previous example, due to the increasing number of elements [4].

Table 2: Average GPU per-step timing (in ms) for the wave propagation in a bulk medium

# MPI	1	2	4	8
Volume	569.93	287.23	142.95	72.41
Surface	10.01	5.04	2.55	1.31
Comm.	0	35.51	33.74	38.79
Per-Step	1,482.00	777.12	403.45	221.94

V. CONCLUSION

The DGTD algorithm was extended to model the instantaneous third-order Kerr-type nonlinearity. The resulting computationally intensive DGTD algorithm was accelerated with GPUs based on the parallelization framework from our prior work. Numerical examples demonstrated that the DGTD simulation was able to capture various nonlinear phenomena and the GPU acceleration was able to achieve a good speedup for this computationally intensive simulation.

REFERENCES

[1] R. W. Boyd, *Nonlinear Optics*. Burlington, MA: Academic Press, 2008.

[2] B. Saleh and M. Tech, *Fundamentals of Photonics*. New York, NY: Wiley, 2013.

[3] R. M. Joseph and A. Taflove, "FDTD Maxwell's equations models for nonlinear electrodynamics and optics," *IEEE Trans. Antennas Propag.*, vol. 45, pp. 364-374, Mar. 1997.

[4] H.-T. Meng and J.-M. Jin, "Acceleration of the dual-field domain decomposition algorithm using MPI-CUDA on large-scale computing systems," *IEEE Trans. Antennas Propag.*, vol. 62, no. 9, pp. 4706-4715, Sept. 2014.

Multilevel Inverse-Based Factorization Preconditioner for Solving Sparse Linear Systems in Electromagnetics

Yiming Bu [1,2], Bruno Carpentieri [3], Zhaoli Shen [1,2], and Tingzhu Huang [2]

[1] Institute of Mathematics and Computer Science, University of Groningen, Groningen, 9712 CP, The Netherlands
yangyangbu@126.com, z.shen@rug.nl

[2] School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu, 611731 China tingzhuhuang@126.com

[3] School of Science and Technology, Nottingham Trent University, Burton Street, Nottingham NG1 4BU, UK
bruno.carpentieri@ntu.ac.uk

Abstract — We introduce an algebraic recursive multilevel approximate inverse-based preconditioner, based on a distributed Schur complement formulation. The proposed preconditioner combines recursive combinatorial algorithms and multilevel mechanisms to maximize sparsity during the factorization.

Index Terms — Approximate inverse preconditioners, computational electromagnetics, Krylov subspace methods, sparse matrices.

I. INTRODUCTION

We consider multilevel approximate inverse-based factorization preconditioners for solving systems of linear equations;

$$Ax = b, \tag{1}$$

where $A \in \mathbb{C}^{n\times n}$ is a typically large nonsymmetric sparse matrix arising from finite difference, finite element or finite volume discretization of systems of partial differential equations in electromagnetism applications. Approximate inverse methods directly approximate A^{-1} as the product of sparse matrices, so that the preconditioning operation reduces to forming one (or more) sparse matrix-vector product(s). Due to their inherent parallelism and numerical robustness, this class of methods are receiving renewed consideration for iterative solutions of large linear systems on emerging massively parallel computer systems. In practice, however, some questions need to be addressed. First of all the computed preconditioner could be singular. In the second place, these techniques usually require more CPU-time to compute the preconditioner than Incomplete LU factorization (ILU)-type methods. Third, the computation of the sparsity pattern of the approximate inverse can be problematic, as the inverse of a general sparse matrix is typically fairly dense. This leads to prohibitive computational and storage costs.

In this paper we present experiments with an algebraic recursive multilevel inverse-based factorization preconditioner that attempts to remedy these problems. The solver, proposed in [1], uses recursive combinatorial algorithms to preprocess the structure of A and to produce a suitable ordering of the unknowns of the linear system that can maximize sparsity in the approximate inverse. An efficient tree-based recursive data structure is generated to compute and apply the multi-level approximate inverse fast and efficiently. We assess the effectiveness of the sparse approximate inverse to reduce the number of iterations of Krylov methods for solving matrix problems arising from electromagnetism applications, also against other popular solvers in use today.

II. THE MULTILEVEL FRAMEWORK

We divide the solution of the linear system into the following five distinct phases:

1) a *scale phase*, where the matrix A is scaled by rows and columns so that the largest entry of the scaled matrix has magnitude smaller than one;
2) a *preorder phase*, where the structure of A is used to compute a suitable ordering that maximizes sparsity in the approximate inverse factors;
3) an *analysis phase*, where the sparsity preserving ordering is analyzed and an efficient data structure is generated for the factorization;
4) a *factorization phase*, where the nonzero entries of the preconditioner are actually computed;
5) a *solve phase*, where all the data structures are accessed for solving the linear system.

A. Scale phase

Prior to solving the system, we scale it by rows and

Submitted On: August 30, 2015
Accepted On: April 11, 2016

columns to reduce its condition number. We replace system (1) with:

$$D_1^{1/2} A y = D_1^{1/2} b, \quad y = D_2^{1/2} x, \tag{2}$$

where the $n \times n$ diagonal scaling matrices have the form:

$$D_1(i,j) = \begin{cases} \dfrac{1}{\max\limits_i |a_{ij}|}, & \text{if i = j} \\ 0, & \text{if i} \neq \text{j} \end{cases},$$

$$D_2(i,j) = \begin{cases} \dfrac{1}{\max\limits_j |a_{ij}|}, & \text{if i = j} \\ 0, & \text{if i} \neq \text{j} \end{cases}.$$

For simplicity, we still refer to the scaled system (2) as $Ax = b$.

B. Preorder phase

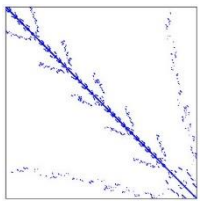
We describe this step using standard notation of graph theory. First, we compute the undirected graph $\Omega(\tilde{A})$ associated with the matrix;

$$\tilde{A} = \begin{cases} A, & \text{if A is symmetric,} \\ A + A^T, & \text{if A is unsymmetric.} \end{cases}$$

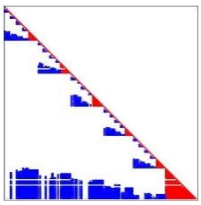
Then, $\Omega(\tilde{A})$ is partitioned into p non-overlapping subgraphs Ω_i of roughly equal size by using the multilevel graph partitioning algorithms available in the Metis package [2]. For each partition Ω_i we distinguish two disjoint sets of nodes: *interior nodes* that are connected only to nodes in the same partition, and *interface nodes* that straddle between two different partitions; the set of interior nodes of Ω_i form a so called *separable* or *independent cluster*. After renumbering the vertices of Ω one cluster after another, followed by the interface nodes as last, and permuting A according to this new ordering, a block bordered linear system is obtained, with coefficient matrix of the form:

$$\tilde{A} = P^T A P = \begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} B_1 & & & F_1 \\ & \ddots & & \vdots \\ & & B_p & F_p \\ E_1 & \cdots & E_p & C \end{pmatrix}. \tag{3}$$

In (3), each diagonal block B_i corresponds to the interior nodes of Ω_i; the blocks E_i and F_i correspond to the interface nodes of Ω_i; the block C is associated to the mutual interactions between the interface nodes. In our multilevel scheme we apply the same block downward arrow structure to the diagonal blocks of $\tilde{A}$ recursively, until a maximum number of levels is achieved or until the blocks at the last level are sufficiently small and easy to factorize. As an example, in Fig. 1 (a) we show the structure of the general sparse matrix *rdb2048* from Tim Davis matrix collection [3] after three reordering levels.



(a) The structure of rdb2048 after permutation



(b) The structure of the inverse factor (In red are displayed the entries actually stored)

Fig. 1. Structure of the multilevel inverse-based factorization for the matrix rdb2048.

C. Analysis phase

The data format for storing the block bordered form (3) of $\tilde{A}$ is defined, allocated and initialized using a tree structure. The root is the whole graph Ω and the leaves at each level are the independent clusters of each subgraph. In other terms, each node of the tree corresponds to one partition Ω_i or equivalently to one block B_i of $\tilde{A}$. The information stored at each node are the entries of the off-diagonal blocks E and F of $B_i's$ father, and those of the block C of B_i after its permutation, except at the last level of the tree where we store the entire block B. These blocks are stored in sparse format.

D. Factorization phase

In this phase, we compute the approximate inverse factors $\tilde{L}^{-1}$ and $\tilde{U}^{-1}$ of $\tilde{A}$, which have the following form:

$$\tilde{L}^{-1} \approx \begin{pmatrix} U_1^{-1} & & & W_1 \\ & \ddots & & \vdots \\ & & U_p^{-1} & W_p \\ & & & U_S^{-1} \end{pmatrix},$$

$$\tilde{U}^{-1} \approx \begin{pmatrix} L_1^{-1} & & & \\ & \ddots & & \\ & & L_p^{-1} & \\ G_1 & \cdots & G_p & L_S^{-1} \end{pmatrix},$$

where $B_i = L_i U_i$, and

$$W_i = -U_i^{-1} L_i^{-1} F_i U_S^{-1}, \quad G_i = -L_S^{-1} E_i U_i^{-1} L_i^{-1}, \tag{4}$$

and L_S, U_S are the triangular factors of the Schur complement matrix:

$$S = C - \sum_{i=1}^{p} E_i B_i^{-1} F_i.$$

During the factorization, fill-in may occur in $\tilde{L}^{-1}$ and $\tilde{U}^{-1}$ but only within the nonzero blocks. Additional sparsity is gained by applying the arrow structure (3) to the diagonal blocks recursively. This can be seen in Fig.

1 (b). For computing the factorization we only need to invert explicitly the last level blocks and the small Schur complements at each reordering level. The blocks W_i, G_i do not need to be assembled. They may be applied using Eq. (4). For the *rdb2048* problem in Fig. 1 (b), we display in red the entries that we actually stored for computing the exact multilevel inverse factorization; these are only 34% of the nonzeros of *A*.

E. Solve phase

In the solve phase, the multilevel factorization is applied at every iteration step of a Krylov method for solving the linear system. Notice that the inverse factorization of $\tilde{A}$ may be written as:

$$(PAP^T)^{-1} = \begin{pmatrix} U^{-1} & W \\ 0 & U_S^{-1} \end{pmatrix} \times \begin{pmatrix} L^{-1} & 0 \\ G & L_S^{-1} \end{pmatrix}, \qquad (5)$$

where $W = -U^{-1}L^{-1}FU_S^{-1}$, $G = -L_S^{-1}EU^{-1}L^{-1}$, and L_S, U_S are the inverse factors of the Schur complement matrix $S = C - EB^{-1}F$.

From Eq. (5), we obtain the following expression for the exact inverse:

$$\begin{pmatrix} B^{-1} + B^{-1}FS^{-1}EB^{-1} & -B^{-1}FS^{-1} \\ -S^{-1}EB^{-1} & S^{-1} \end{pmatrix}. \qquad (6)$$

We can derive preconditioners from Eq. (6) by computing approximate solvers $\tilde{B}^{-1}$ for B and $\tilde{S}^{-1}$ for S. Hence, the preconditioner *M* has the form:

$$M = \begin{pmatrix} \tilde{B}^{-1} + \tilde{B}^{-1}F\tilde{S}^{-1}E\tilde{B}^{-1} & -\tilde{B}^{-1}F\tilde{S}^{-1} \\ -\tilde{S}^{-1}E\tilde{B}^{-1} & \tilde{S}^{-1} \end{pmatrix}.$$

III. NUMERICAL EXPERIMENTS

We show some preliminary results with the proposed Algebraic Multilevel Explicit Solver (AMES) for solving a set of matrix problems arising from electromagnetics applications [3]. We summarize the list of problems in Table 1. In our experiments, we choose ILUPACK [7] as the local solver in AMES to invert the diagonal blocks at the last level, and the Schur complements at each level. Notice that in this case the entries of the inverse factors are not computed explicitly, and the application of the preconditioner is carried out through a backward and forward substitution procedure. We solve the right preconditioned system $AMy = b$, $x = My$ instead of (1), using restarted GMRES [4] preconditioned by AMES. We compare AMES against two other popular algebraic preconditioners for linear systems, that are the Algebraic Recursive Multilevel Method (ARMS) by Saad and Suchomel [5] and the Sparse Approximate Inverse preconditioner (SPAI) by Grote and Huckle [6], at roughly equal memory costs.[1] We use the zero vector as initial guess in our code, and we terminate the solution process when the norm of residual is below 10^{-12} or the iterations count exceeds 5000. For the performance comparison, we report on the memory ratio $\frac{nnz(M)}{nnz(A)}$, number of iterations (*Its*), and time costs for performing the preordering phase (t_p), the factorization phase (t_f) and the solving phase (t_s). The experiments are run in double precision floating point arithmetic in Fortran95, on a PC equipped with an Intel(R) Core(TM) i5-3470 running at 3.20 GHz and with 8 GB of RAM and 6144 KB of cache memory.

Table 1: Set and characteristics of test matrix problems

Matrix Problem	Size	*nnz*(*A*)	Field
dw2048	2,048	10,114	Square dielectric waveguide
dw8192	8,192	41,746	Square dielectric waveguide
utm3060	3,060	42,211	Uedge test matrix
utm5940	5,940	83,842	Uedge test matrix
2cubes_sphere	101,492	874,378	FEM electromagnetics

A. Varying number of reduction levels in AMES

We consider the *dw2048*, *dw8192* and *2cubes_sphere* problems for these experiments. Increasing the number of levels may help reduce the number of iterations at similar memory cost. In our experiments, varying the number of levels n_{lev} from 1 to 3 for a given problem, we tuned the dropping threshold to keep roughly the same memory cost in each run, and then we studied the effect on convergence. The results of our experiments, reported in Table 2, show that using more levels enabled us to reduce the number of iterations at similar memory ratio. However, the computing time for the preordering phase (t_p) and the solution cost per iteration tend to increase with the n_{lev}. We conclude that a small number of reduction levels is recommended to use in AMES.

Table 2: Performance of AMES with varying numbers of reduction levels

Matrix	n_{lev}	$\frac{nnz(M)}{nnz(A)}$	*Its*	t_p (sec)	t_f (sec)	t_s (sec)	t_{tot} (sec)
dw2048	1	2.37	24	0.023	0.025	0.008	0.056
	2	2.33	22	0.029	0.021	0.011	0.061
	3	2.38	17	0.030	0.021	0.027	0.078
dw8192	1	3.22	87	0.067	0.109	0.312	0.488
	2	3.27	82	0.083	0.128	0.417	0.628
	3	3.28	78	0.092	0.141	0.744	0.977
2cubes_sphere	1	0.31	12	1.271	3.691	0.310	5.272
	2	0.31	12	1.503	2.552	0.598	4.653
	3	0.31	11	2.333	1.829	1.200	5.362

[1] We choose a combination of parameters for AMES, and tune the dropping threshold for ARMS and SPAI to obtain similar memory cost.

B. Varying the number of reduction levels for the Schur complement

The Schur complement matrix S relative to the block C in (3) typically preserves a good deal of sparsity that can be exploited during the factorization by reordering S in a multilevel nested dissection structure, similarly to what is done to the upper leftmost block B. We have implemented this idea at the first permutation level, using ILU factorization as local solver for the reduced Schur complement matrix. We denote by AS_{lev} the number of reduction levels used for the Schur complement. We consider again the *dw2048*, *dw8192* and *2cubes_sphere* problems in these experiments. For a certain test problem, we vary AS_{lev} keeping all the other parameters constant, and we tune the drop tolerance in the ILU factorization to have similar memory costs. The value $AS_{lev} = 0$ means that only the diagonal blocks of the upper-left block B are permuted. Clearly, the max value of AS_{lev} is limited by the size of Schur complement. From Table 3, we see that simultaneous permutation of both the diagonal blocks of B and of the Schur complement S can make the AMES solver more robust to some extent. However, the implementation cost increases and thus, although useful, this option is problem dependent. In our experiments of the coming sections, we select the value for the parameter AS_{lev} that minimizes the total solution cost.

Table 3: Performance of AMES with varying numbers of reduction levels

Matrix	AS_{lev}	$\frac{nnz(M)}{nnz(A)}$	*Its*	t_p (sec)	t_f (sec)	t_s (sec)	t_{tot} (sec)
dw2048	0	2.37	24	0.023	0.025	0.008	0.056
	1	2.37	12	0.023	0.027	0.005	0.055
	2	2.37	12	0.024	0.031	0.012	0.067
dw8192	0	3.22	87	0.067	0.109	0.312	0.488
	1	3.26	21	0.067	0.164	0.057	0.288
	2	3.26	18	0.073	0.156	0.060	0.289
2cubes_ sphere	0	0.31	12	1.271	3.691	0.310	5.272
	1	0.31	11	1.277	3.974	0.334	5.585
	2	0.31	11	1.288	4.016	0.350	5.654
	3	0.31	11	1.298	3.985	0.355	5.638

C. Comparing AMES against other preconditioners

From Table 4, we can clearly see that the AMES preconditioner shows a good potential of reducing the number of iterations against other state-of-the-art preconditioning techniques at similar memory costs. This result demonstrates the overall good efficiency of the fill reducing strategies implemented in the preconditioner on the selected electromagnetic problems. One exception is the 2cubes_sphere problem, which has favourable properties for the SPAI method. The good decay of the entries away from the diagonal makes this problem suitable for SPAI. The AMES method still remains competitive. However, the pre-processing and solution costs for setting up and applying the multilevel recursive scheme do not pay off in this case.

Table 4: Performance comparison of the multilevel approximate inverse preconditioner against other iterative solvers

Matrix	Method	$\frac{nnz(M)}{nnz(A)}$	*Its*	t_p (sec)	t_f (sec)	t_s (sec)	t_{tot} (sec)
dw2048	AMES	2.37	12	0.023	0.027	0.005	0.055
	ARMS	2.39	670	0	0.009	0.081	0.090
	SPAI	2.37	2239	0	0.094	0.367	0.461
dw8192	AMES	3.26	21	0.067	0.164	0.057	0.288
	ARMS	3.37	+5000	0	0.040	+10.89	+10.93
	SPAI	3.33	+5000	0	0.836	+4.841	+5.677
Utm3060	AMES	2.79	125	0.077	0.145	0.366	0.588
	ARMS	2.93	402	0	0.030	0.763	0.793
	SPAI	2.88	+5000	0	3.131	+3.095	+6.226
Utm5940	AMES	3.50	267	0.147	0.409	2.738	3.294
	ARMS	3.51	1150	0	0.077	5.085	5.162
	SPAI	3.51	+5000	0	11.76	+11.02	+22.78
2cubes_ sphere	AMES	0.31	12	1.271	3.691	0.310	5.272
	ARMS	0.32	68	0	0.262	0.986	1.248
	SPAI	0.32	8	0	3.269	0.153	3.422

IV. CONCLUSIONS

In this paper we used recursive combinatorial techniques to remedy two typical drawbacks of explicit preconditioning, that are lack of robustness and high construction cost. The numerical experiments show that these strategies can improve the performance of conventional approximate inverse methods, yielding iterative solutions that can compete favourably against other popular solvers in use today.

REFERENCES

[1] Y. Bu, B. Carpentieri, Z. Shen, and T.-Z. Huang, "A hybrid recursive multilevel incomplete factorization preconditioner for solving general linear systems," *Applied Numerical Mathematics*, vol. 104, pp. 141-157, 2016.

[2] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput*., vol. 20, pp. 359-392, 1999.

[3] T. Davis, *Sparse Matrix Collection*, (1994). Available at the URL: http: //www.cise.ufl.edu/ research/sparse/matrices

[4] Y. Saad, *Iterative Methods for Sparse Linear Systems*. SIAM Publications, 2nd edition, 2003.

[5] Y. Saad and B. Suchomel, "ARMS: An algebraic recursive multilevel solver for general sparse linear systems," *Numer. Linear Algebra Appl*., vol. 9, no. 5, pp. 359-378, 2002.

[6] M. Grote and T. Huckle, "Parallel preconditionings with sparse approximate inverses," *SIAM J. Sci. Comput*., vol. 18, pp. 838-853, 1997.

[7] M. Bollhoefer, Y. Saad, and O. Schenk, *ILUPACK - Preconditioning Software Package*, 2010. Available online at the URL: http://ilupack.tu-bs.de.

Porting an Explicit Time-Domain Volume Integral Equation Solver onto Multiple GPUs Using MPI and OpenACC

Saber Feki [1], Ahmed Al-Jarro [3], and Hakan Bagci [2]

[1] KAUST Supercomputing Laboratory
[2] Division of Computer, Electrical and Mathematical Sciences and Engineering
King Abdullah University of Science and Technology (KAUST), Thuwal, 23955-6900, KSA
{saber.feki, hakan.bagci}@kaust.edu.sa

[3] Department of Electronic and Electrical Engineering
University College London, Torrington Place, WC1E 7JE, London, UK
ahmed.aljarro@ucl.ac.uk

Abstract ― A scalable parallelization algorithm to port an explicit marching-on-in-time (MOT)-based time domain volume integral equation (TDVIE) solver onto multi-GPUs is described. The algorithm makes use of MPI and OpenACC for efficient implementation. The MPI processes are responsible for synchronizing and communicating the distributed compute kernels of the MOT-TDVIE solver between the GPUs, where one MPI task is assigned to one GPU. The compiler directives of the OpenACC are responsible for the data transfer and kernels' offloading from the CPU to the GPU and their execution on the GPU. The speedups achieved against the MPI/OpenMP code execution on multiple CPUs and parallel efficiencies are presented.

Index Terms ― Explicit marching-on-in-time scheme, GPU, MPI, OpenACC, time-domain volume integral equation.

I. INTRODUCTION

The use of hardware accelerators, including multi and many-core architectures, has been increasing in many emerging applications of high performance computing (HPC) as they provide cost effectiveness, power efficiency, and physical density. Nevertheless, one of the limiting factors to a wider spread use of multi-core accelerators, such as GPUs, is the human-labor intensive porting process required by low-level programming models, such as CUDA [1] and OpenCL [2]. To overcome this limit, HPC research has focused on developing high-level directive based programming models, such as OpenACC [3], which provide compiler directives and clauses to annotate codes originally developed for CPUs in a manner similar to how OpenMP [4] is used on codes executed on multicore CPU architectures. This high-level approach, when carefully implemented, significantly reduces the re-programming efforts while maintaining the efficiency of the resulting codes.

In this work, we report on our recent efforts on parallelizing a fully explicit marching-on-in-time (MOT)-based time-domain volume integral equation (TDVIE) solver [5] for efficient execution on multiple GPUs. The MOT-TDVIE solvers are becoming attractive alternatives to finite difference time domain (FDTD) schemes for analyzing transient electromagnetic scattering from inhomogeneous dielectric objects [5, 6]. However, their effective use in practical problems of photonics, optoelectronics, and bio-electromagnetics, where electrically large scatterers need to be discretized with millions of degrees of freedom, relies on acceleration algorithms such as the plane-wave time domain (PWTD) method [7] and/or hardware-based acceleration [8-11].

Our recent research has focused on the latter; we developed highly scalable parallelization algorithms [8, 9] to enable the explicit MOT-TDVIE solver of [5] in analyzing scattering from electrically large structures. Additionally, we used OpenACC to enable the execution of the same solver on GPUs [10, 11]. Significant performance improvements with up to 30X and 11X speedups relative to the sequential and multi-threaded CPU codes were achieved. Furthermore, we demonstrated that the (single) GPU-accelerated MOT-TDVIE solver could leverage energy consumption gains on the order of 3X relative to its multi-threaded CPU version [10]. In this paper, we describe in detail the process of porting the same MOT-TDVIE solver onto *multi*-GPUs using MPI/OpenACC. Additionally, we present numerical results, which demonstrate that the ported code executes up to 11.2X faster on multi-GPUs than on conventional CPUs.

Submitted On: September 20, 2015
Accepted On: July 4, 2016

II. MOT-TDVIE SOLVER

A. MOT-TDVIE algorithm

Let V represent the volumetric support of dielectric scatterer with permittivity $\varepsilon(\mathbf{r})$ residing in an unbounded background medium with permittivity ε_0. The scatterer is excited by a band-limited incident electric field $\mathbf{E}_0(\mathbf{r},t)$. Upon excitation, scattered field $\mathbf{E}^{\text{sca}}(\mathbf{r},t)$ is generated. Scattered and incident fields satisfy $\mathbf{E}(\mathbf{r},t)=\mathbf{E}_0(\mathbf{r},t)+\mathbf{E}^{\text{sca}}(\mathbf{r},t)$, where $\mathbf{E}(\mathbf{r},t)$ is the unknown "total" field. One can construct a TDVIE as [5, 6]:

$$\mathbf{E}(\mathbf{r},t)=\mathbf{E}_0(\mathbf{r},t)+\left[\nabla\nabla\cdot-\partial_t^2/c_0\right]\mathbf{A}(\mathbf{r},t),\ \mathbf{r}\in V, \quad (1)$$

where $\mathbf{A}(\mathbf{r},t)$ is given by:

$$\mathbf{A}(\mathbf{r},t)=\int_V\frac{(\varepsilon(\mathbf{r}')-\varepsilon_0)\mathbf{E}(\mathbf{r}',t-R/c_0)}{4\pi\varepsilon_0 R}dv',\ \mathbf{r}'\in V. \quad (2)$$

Here, c_0 is the speed of light in the background medium, and $R=|\mathbf{r}-\mathbf{r}'|$ is the distance between points $\mathbf{r}$ and $\mathbf{r}'$. TDVIE (1) is solved by time marching, which makes use of an explicit predictor-corrector algorithm as described next [5]. First V is discretized using N_e cubic elements. Let $\mathbf{r}_k$, $k=1:N_e$, and Δt represent the centers of these elements and time step size. Assume n represents the index of the "current" time step. At the predictor step, first $\mathbf{A}_{k,n}=\mathbf{A}(\mathbf{r}_k,n\Delta t)$ are computed using $\mathbf{E}_{l,m}=\mathbf{E}(\mathbf{r}_l,m\Delta t)$, $l=1:N_e$, $m=\max(1,n-N_g):n\Delta t$, in the integral given in (2). For this operation, $\mathbf{E}(\mathbf{r},m\Delta t)=\mathbf{E}_{l,m}$ is assumed within cubic element l and linear interpolation is used to approximate $\mathbf{E}(\mathbf{r}_l,n\Delta t-R_{kl}/c_0)$, where $R_{kl}=|\mathbf{r}_k-\mathbf{r}_l|$, from $\mathbf{E}_{l,m-1}$ and $\mathbf{E}_{l,m}$ for $[m-1]\Delta t<n\Delta t-R_{kl}/c_0<m\Delta t$. Note that here $N_g=\lfloor R_{\max}/c_0\Delta t\rfloor+2$, where $R_{\max}=\max\{R_{kl}\}$, for any $\mathbf{r}_k,\mathbf{r}_l\in V$. Then, finite differences (FD), which approximate the spatial derivative operator "$\nabla\nabla\cdot$", are applied to $\mathbf{A}_{k,n}$ to yield "predicted" samples $\mathbf{E}_{k,n}$. Differentiation "∂_t^2" in (1) is approximated using backward FD for pairs $(\mathbf{r}_k,\mathbf{r}_l)$ that satisfy $R_{kl}<2c_0\Delta t$ and using central FD for all other pairs. At the corrector step, differentiation "∂_t^2" is *recomputed* using a central difference formula for pairs $(\mathbf{r}_k,\mathbf{r}_l)$ that only satisfy $R_{kl}<2c_0\Delta t$. Note that use of central FD is now allowed since field samples that are not known at the predictor step (due to causality) can now be replaced by the predicted fields' samples. At the end of time step n, $\mathbf{E}_{k,m}$ are stored as part of the "history" of field samples to be used in the computation of $\mathbf{A}_{k,n+1}$.

Note that FD evaluations and corrector updates are spatially local operations while computation of $\mathbf{A}_{k,n}$, $k=1:N_e$, is global. Samples $\mathbf{E}_{l,m}$ that satisfy the condition $[n-m]c_0\Delta t>R_{kl}$ do not contribute to $\mathbf{A}_{k,n}$ since the fields radiated from point $\mathbf{r}_l$ at time $m\Delta t$ have not yet reached point $\mathbf{r}_k$ at time $n\Delta t$. This also means that for $n\geq N_g$, all fields radiated from all points reach to all other points. Consequently, they all contribute to all samples $\mathbf{A}_{k,n}$, $k=1:N_e$, rendering the computational cost of the integral evaluation $O(N_e^2)$ per time step for all $n\geq N_g$. As N_e increases, the cost of computing $\mathbf{A}_{k,n}$ limits the solver's applicability to electrically large problems. This limitation can be overcome by using acceleration algorithms such as the PWTD method [6-7] and/or highly scalable parallelization algorithms [8-11]. In this work, we implement and fine-tune the parallelization algorithm of [8, 9], which is originally developed for CPUs, for multi-GPUs to further increase the applicability of the MOT-TDVIE solver to electrically large problems.

B. MPI parallelization

Operations required by the MOT-TDVIE solver at each time step can be grouped into two: (i) computation of $\mathbf{A}_{k,l}$, $k=1:N_e$, which requires access to samples $\mathbf{E}_{l,n-m}$, $l=1:N_e$, $m=1:\min(n-1,N_g)$ and (ii) computation of samples $\mathbf{E}_{k,n}$ by applying FD to $\mathbf{A}_{k,n}$. The parallelization scheme used here, first, ensures the even distribution of the memory via application of the graph-based partitioning scheme to the distribution of the points $\mathbf{r}_k$, $k=1:N_e$, representing the discretization of V. This results in an unstructured partitioning of the points $\mathbf{r}_k$ [9]. In this partitioning, each process stores only $\mathbf{E}_{k,n}$ and $\mathbf{E}_{l,n-m}$ that belong to the partition assigned to it. The computational load of step (i) is distributed using a one-way pipeline communication strategy, so-called the "rotating tiles" paradigm [8]. The test tiles (partitions that contain test points) are initially same as the source titles (partitions that contain source points) at the beginning of the rotation but they are rotated among the processors during the computation of $\mathbf{A}_{k,l}$. When a processor receives a test tile, it first adds the contribution from the source titles it stores to $\mathbf{A}_{k,l}$ associated with the received test tile, then it passes the (updated) tile to its "neighboring" processor. At the end of a full rotation all contributions to $\mathbf{A}_{k,l}$ are computed. It is noted here that, this strategy eliminates the need for globally executed collective routines such as MPI_Reduce [8]. The computational load of step (ii) is distributed using the same grouping of the test points provided by the graph-partitioning algorithm. This reduces the communication costs associated with spatial FD computations by ensuring that the data communication only happens between points residing on the boundary of any two partitions [9].

III. PORTING TO MULTIPLE GPUS

The state-of-the-art GPU-nodes can include up to 8 K80 GPUs, which is essentially equivalent to having 16 independent GPUs. On the other hand, OpenACC standard, as a stand-alone programming model, provides very limited support for code development on multiple devices. Therefore, one typically relies on using OpenACC/OpenMP together with the MPI standard to port codes onto a cluster of nodes equipped with multiple GPUs/multicore CPUs. In this work, OpenACC is used to accelerate the time marching loop of the MOT-TDVIE

solver. Both memory- and compute-bound operations are executed on GPUs, which benefit from the improved memory bandwidth and higher flop rate, respectively. However, because the amount of compute-bound operations is significantly higher that memory-bound operations, benefits from increased memory bandwidth might be considered negligible. The main advantages of OpenACC over CUDA are the significantly increased programming efficiency and code portability on different hardware platforms. More specifically, OpenACC offers an easy way to port codes onto accelerators using simple descriptive compiler directives. Additionally, the same OpenACC-annotated code can be compiled on different hardware platforms, including the host itself (multicore CPU architecture) as well as any other accelerator supported by the OpenACC standard. In contrast, the CUDA programming model is more tedious to implement and can be used on only NVIDIA GPUs.

The code is designed such that the number of MPI processes spawn on each node is equal to the number of GPUs per node. Each MPI process is assigned to a GPU using the runtime API function *acc_set_device_num* to set the GPU target to the MPI rank modulo the number of GPUs per node, as shown in the pseudo code in Fig. 1. The data directive *#pragma acc data* is applied to the outermost time loop in order to minimize data transfers between the host and the device. Input and output arrays are annotated with clauses *present_or_copyin* and *present_or_copyout*, respectively. However, the arrays needed for the MPI communications, which are of very limited memory size, are copied in and out at each iteration so that they are accessible to the MPI routines. Each enclosed code block in the MOT-TDVIE solver is annotated with *#pragma acc kernels* and offloaded to the assigned GPU. The code blocks implementing the computation of $\mathbf{A}_{k,l}$ consist of two nested loops yielding a quadratic computational complexity. The second loop is further annotated with *#pragma acc loop reduction* and the associated variables to further optimize the sum operation of all source contributions. The OpenACC standard offers the ability to further tune loop execution using the *gang* and *vector* clauses, which can be used to modify the number of blocks of threads and threads per block to be executed, respectively. Since there are only two nested loops in the kernels of the parallel MOT-TDVIE solver, values assigned to these two parameters by the compiler already result in good performance improvements. Having said that, tuning these parameters in the presence of three or more nested loops may significantly increase the performance. Indeed, this was demonstrated for the serial version of the code, with structured grid, when executed on single GPUs. The tuning of these two parameters improved the acceleration performance by up to 23X [10]. For some of the loops that are not parallelized by the compiler due to perceived false data dependencies, the code block is annotated with the loop pragma accompanied with the independent clause to avoid unnecessary synchronization between the loop iterations in absence of data dependencies. Note that, the code design using multi-GPU kernels allows for MPI synchronizations and communications to take place between the compute kernels as necessary. That is the case with the rotating tiles communications implemented to compute $\mathbf{A}_{k,l}$, and the halo cells exchange communications implemented to compute $\mathbf{E}_{k,n}$ using FD.

```
// Get number of MPI processes = # of GPUs
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
// Assign each MPI process to a GPU
acc_set_device_num(rank%ngpus,acc_device_nvidia);

#pragma acc data present_or_copyin(input
arrays) present_or_copyout(output arrays)

for (int t=0;  t<nt; ++t) {  // time loop
  for(rot=0; rot<=size; ++rot){
    // MPI communication for rotating tiles
    MPI_Sendrecv();
    MPI_Barrier();
    #pragma acc kernels
    // Spatio-temporal convolutions
    for (int k=0; k<Ne; ++k){
      #pragma acc loop reduction
      for (int l=0; l<Ne; ++l){
          A[t][k] = A[t-tkl][l] + ...
  }  }
  MPI_Barrier();
    // Loops with no data dependencies
    #pragma acc kernels
    #pragma acc loop independent
    for(i=0; i<ni; ++i){

    }
  } // end rotation

  // MPI communication for Halo Exchange
  MPI_Sendrecv();
  MPI_Barrier();
  #pragma acc kernels
  // spatial finite difference operations
  for (int k=0; k<Ne; ++k){
    B[t][k] = A[t][k] + ....
  }
} // end time loop
```

Fig. 1. Pseudo code for the implementation of the MOT-TDVIE solver using MPI and OpenACC.

IV. NUMERICAL EXPERIMENTS

The test bed used for performance evaluation consists of a system of two nodes connected using an Infiniband FDR high-speed network. Each node is a dual socket CPU system hosting four NVIDIA Kepler K20c GPUs. Each socket is an eight-core Sandy Bridge

Intel(R) Xeon(R) CPU E5-2650.

In our performance evaluation, as shown in Fig. 2, a significant speedup ranging from 7.4X to 11.2X is recorded comparing the MPI and OpenMP implementation on 16 cores SandyBridge to the MPI and OpenACC implementation on four K20c GPUs. It is also observed that as N_e increases, higher speedup is achieved. This is due to the fact that the GPUs are supplied with larger computational loads; therefore, taking better advantage of its computational capacity. It has been shown before that the MPI implementation demonstrated a great scalability on large super-computers [8-9]. Figure 3 shows the parallel efficiency of the MPI and OpenACC implementation executed on two and eight GPUs, which ranges from 82% to 94%. Another advantage of using NVIDIA GPUs is their energy efficiency as the simulation consumed 2.4X less energy on GPUs than on CPUs. For all of the above, the GPUs are identified as the preferred computing platform in our overall performance analyses of the explicit MOT-TDVIE solver.

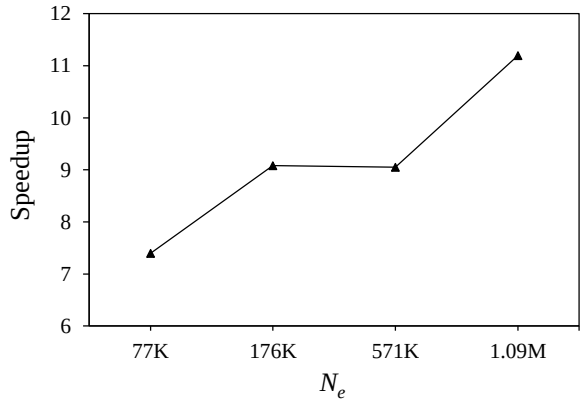


Fig. 2. Performance speedup of MPI and OpenACC on four K20c GPUs compared to MPI and OpenMP on 16 cores SandyBridge CPU.

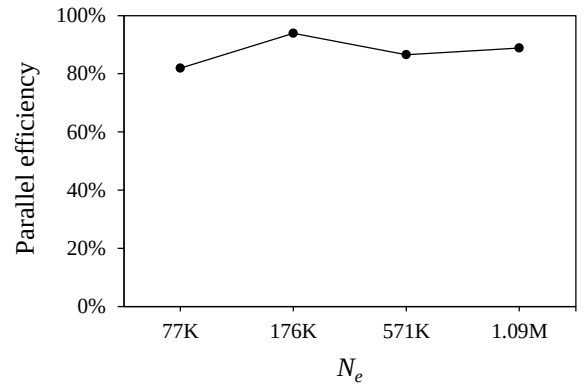


Fig. 3. Parallel efficiency of the MPI and OpenACC implementation scaling from two to eight GPUs.

V. CONCLUSION

The porting of the explicit MOT-TDVIE solver using MPI and OpenACC to multi-GPUs resulted in a highly efficient implementation. The simulations executed on multi-GPUs were faster by up to an order of magnitude compared to those executed on CPUs (using the MPI and OpenMP version of the code). The OpenACC API has the advantage of easily porting the MPI code to multi-GPU environment; therefore, increases the developer productivity while keeping the legacy of the original CPU code. Furthermore, the parallelization allows the explicit TDVIE solver to efficiently simulate transient electromagnetic wave interactions on electrically large structures discretized using a large number of spatial elements on GPUs.

REFERENCES

[1] CUDA, www.nvidia.com, 2016.

[2] OpenCL, www.khronos.org/opencl, 2016.

[3] OpenACC, www.openacc-standard.org, 2016.

[4] OpenMP, www.openmp.org, 2016.

[5] A. Al-Jarro, M. A. Salem, H. Bagci, T. M. Benson, P. Sewell, and A. Vukovic, "Explicit solution of the time domain volume integral equation using a predictor-corrector scheme," *IEEE Trans. Antennas Propag.*, vol. 60, no. 11, pp. 5203-5214, 2012.

[6] N. T. Gres, A. A. Ergin, E. Michielssen, and B. Shanker, "Volume-integral-equation-based analysis of transient electromagnetic scattering from three-dimensional inhomogeneous dielectric objects," *Radio Sci.*, vol. 36, no. 3, pp. 379-386, May 2001.

[7] Y. Liu, A. Al-Jarro, H. Bagcı, and E. Michielssen, "Parallel PWTD-accelerated explicit solution of the time domain electric field volume integral equation," *IEEE Trans. Antennas Propag.*, vol. 64, no. 6, pp. 2378-2388, 2016.

[8] A. Al-Jarro, M. Cheeseman, and H. Bagci, "A distributed-memory parallelization of the explicit time-domain volume integral equation solver using a rotating tiles paradigm," in Proc. *28th Int. Review of Progress in Appl. Comp. Electromagn.*, 2012.

[9] A. Al-Jarro and H. Bagci, "An unstructured mesh partitioning scheme for efficiently parallelizing an explicit time domain volume integral equation solver," in Proc. *29th Int. Review of Progress in Appl. Comp. Electromagn.*, 2013.

[10] S. Feki, A. Al-Jarro, A. Clo, and H. Bagci, "Porting an explicit time-domain volume-integral-equation solver on GPUs with OpenACC," *IEEE Antennas Propag. Mag.*, vol. 56, pp. 265-277, 2014.

[11] S. Feki, A. Al-Jarro, and H. Bagci, "Multi-GPU-based acceleration of the explicit time domain volume integral equation solver using MPI-OpenACC," in Proc. *IEEE Int. Symp. Antennas Propag. and USNC/URSI National Radio Sci.*, Meet., 2013.

Parallel Realization of Element by Element Analysis of Eddy Current Field Based on Graphic Processing Unit

Dongyang Wu, Xiuke Yan, Renyuan Tang, Dexin Xie, and Ziyan Ren

Department of Electrical Engineering
Shenyang University of Technology, Liaoyang, Liaoning 110870, China
shineast_521@163.com, yanxke@126.com, sgdtds@sina.com, xiedx2010@163.com, and rzyhenan@163.com

Abstract ─ The element by element parallel finite element method (EbE-PFEM) applied to engineering eddy current problem is presented in this paper. Unlike classical finite element method (FEM), only element matrix is needed to store for EbE method. Thereby more storage memory saved. Element by element conjugated gradient (EbE-CG) method is used to solve the equations which are discretized from elements level. Considering the ill-conditioned character of system equations, highly parallel Jacobi preconditioned (JP) method is used to accelerate the convergence. Besides, the process of dealing with boundary condition based on EbE theory is introduced. To validate the method, a 2D eddy current problem in complex frequency domain is used. The numerical analysis is carried out on the graphic processing units (GPU) with a compute unified device architecture (CUDA) parallel programming model to accelerate the convergence. And the results demonstrate that the JP method and GPU platform are effective in solving eddy current field with improved convergence.

Index Terms ─ Eddy current filed, element by element method, graphic processing unit, parallel computing.

I. INTRODUCTION

Due to the computer resource requirements of classical FEM for solving the electromagnetic problems, the parallel finite element method (PFEM) has become increasingly popular in recent years. Element by element (EbE) method [1] is a PFEM which can execute the parallelism on the elements level. The advantage of EbE method compared to classical FEM is that it does not need assembling and storing system matrix. Its key idea is to decouple the element solution by directly solving element equations instead of whole equations. The solving process is executed in parallel, and only intermittent communication is needed. Initially, EbE method was used for heat conduction problem and then expanded to the field of mechanics. More recently however, with the development of general purpose on graphic processing unit (GPGPU), EbE method has received increasing attention as it is very suitable for parallel processing and with the GPU[2]-[4] being a multi-core device, parallel processing at element level on different cores can be achieved. Some good results have been obtained with electrostatic problem, as in [5], [6].

In author's previous work, firstly EbB-CG method is directly used to solve 2D eddy current problem parallelly on the GPU, and 3.4 times speed up rate achieved compared with that of serial calculation with CPU [7]. Furthermore, TEAM problem 7 is taken as an example to validate the EbE method and GPU are effective for 3D linear eddy current problem, and the results have a good agreement with experiment data [8]. The purpose of this paper is to broaden the JP method to 2D eddy current analysis with two different medium in solving domain, and a comparing analysis is fulfilled between EbE-CG method and EbE-JPCG method.

II. EBE METHOD AND GPU IMPLEMENTATION

A. Node connection matrix

The key function of node connection matrix (NCM) is to transit the node information between local variables and global variables.

Now, assume $\boldsymbol{x}$ is global solution vector (GSV), $\boldsymbol{x}^e$ is the local elements solution vector (LESV), $\boldsymbol{x}^{(e)}$ is global elements solution vector (GESV), E is the total number of elements, $\boldsymbol{Q}$ is NCM. Then consider three type operations of NCM as below:

$$\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{x}^{(e)}, \tag{1}$$

where $\boldsymbol{x}^{(e)} = \left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(E)}\right)^T$, this operation achieves the alternation from GSV to GESV according to the node number of each element:

$$\boldsymbol{Q}^T \boldsymbol{x}^e = \boldsymbol{x}, \tag{2}$$

where $\boldsymbol{x}^e = \left(\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^E\right)^T$, this operation achieves the summation of LESV which have the same node number. This process alternates the LESV to GSV:

$$\boldsymbol{Q}\boldsymbol{Q}^T \boldsymbol{x}^e = \boldsymbol{x}^{(e)}. \tag{3}$$

Submitted On: November 16, 2015
Accepted On: May 10, 2016

Equation (3) achieves the alternation from LESV to GESV.

NCM also can be operated with the element matrix $\boldsymbol{K}^e$, and the relationship between system matrix $\boldsymbol{K}$ and the element matrix $\boldsymbol{K}^e$ can be given as follows:

$$\boldsymbol{K} = \boldsymbol{Q}^T \boldsymbol{K}^e \boldsymbol{Q}. \tag{4}$$

Equations (1) to (4) provide the theoretical foundation for fulfilling the parallel EbE technique.

B. EbE-CG method

For the traditional FEM, the system matrix $\boldsymbol{K}$ and right hand side (RHS) vector $\boldsymbol{b}$ must be assembled from the element matrix $\boldsymbol{K}^e$ and element RHS $\boldsymbol{b}^e$, while for the EbE-PFEM, considering (1)-(4) the assemble process can be deduced as follows:

$$\boldsymbol{b} = \boldsymbol{Q}^T \boldsymbol{b}^e = \boldsymbol{K}\boldsymbol{x} = \boldsymbol{Q}^T \boldsymbol{K}^e \boldsymbol{Q}\boldsymbol{x} = \boldsymbol{Q}^T \boldsymbol{K}^e \boldsymbol{x}^{(e)}. \tag{5}$$

As shown in (5), the product of assembling the element vector is equivalent with the product of assembling element matrix. So, we can solve the element equations parallelly as below:

$$\boldsymbol{K}^e \boldsymbol{x}^{(e)} = \boldsymbol{b}^e. \tag{6}$$

As we know, CG method mainly contains two types of inner product calculations, i.e., $(\boldsymbol{r},\boldsymbol{r})$ and $(\boldsymbol{p},\boldsymbol{Ap})$ which can be calculated by EbE method as follows:

$$(\boldsymbol{r},\boldsymbol{r}) = \boldsymbol{r}^T\boldsymbol{r} = (\boldsymbol{r}^e)^T \boldsymbol{Q}\boldsymbol{Q}^T \boldsymbol{r}^e = \sum (\boldsymbol{r}^e)^T \boldsymbol{r}^{(e)}, \tag{7}$$

where $\boldsymbol{r}^{(e)} = \boldsymbol{r}^e \oplus \sum_{j \in adj(e)} \boldsymbol{r}^j$, $\boldsymbol{r}$ is the global residual, $\boldsymbol{r}^e$ is the local element residual, $\boldsymbol{Q}$ is the NCM. $\boldsymbol{r}^{(e)}$ is the sum of $\boldsymbol{r}^e$ and $\boldsymbol{r}^j$ which are relative with $\boldsymbol{r}^e$. So this process needs the solution information of adjacent nodes. The calculation of $(\boldsymbol{p},\boldsymbol{Ap})$ is similar with $(\boldsymbol{r},\boldsymbol{r})$.

C. Dealing with boundary condition

It is not necessary to assemble the system matrix for EbE method, so the boundary condition (BC) has to be applied on the elements level. Now, taking an example of 2D with triangular subdivision (Fig. 1), and assume the value of first kind BC is $\boldsymbol{U}_0$.

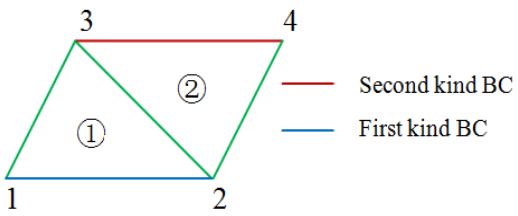


Fig. 1. Partial subdivision of 2D model.

Based on traditional FEM idea, we can get the element matrix equation of ①, as described in (8):

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}. \tag{8}$$

Differ from classical FEM, the element matrix and right hand side vector must be modified with weights simultaneously. Taking element ① as an example, we can get the modified element Equation (9):

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \dfrac{K_{22}^1}{K_{22}^1 + K_{22}^2} & 0 \\ 0 & 0 & K_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \begin{bmatrix} U_0 \\ \dfrac{K_{22}^1}{K_{22}^1 + K_{22}^2} U_0 \\ b_3^1 \end{bmatrix}. \tag{9}$$

In contrast to first kind BC, the second kind BC (node 3 and 4) can be applied on elements directly. For the 2D eddy current problem, the current density is easily applied to the elements level during the element analysis of RHSV.

III. NUMERICAL EXPERIMENT

In this work, a conductor in an open slot of motor is taken as an example to analyze the skin effect. Two models are considered to verify the validity of the proposed method. Model I is shown in Fig. 2, it is a current-carrying conductor in an open slot, for which the analytical solution is available [9], and the domain contains only one conducting medium. And its mathematical model is shown as below:

$$\begin{cases} \dfrac{\partial^2 A}{\partial y^2} = \mathrm{j}\omega\sigma\mu_0 A - \mu_0 J_e = p^2 A - \mu_0 J_e \quad in\,\Omega \\ \dfrac{\partial A}{\partial n} = 0 \quad (on\,AB, CD, BD) \\ \dfrac{\partial A}{\partial y} = \mu_0 H_x = -\dfrac{\mu_0 I_{\mathrm{m}}}{b} \quad (on\,AC) \end{cases}, \tag{10}$$

where $\boldsymbol{A}$ is vector magnetic potential, ω is angular frequency, σ electrical conductivity, μ_0 is magnetic conductivity, J_e is electrical current density, Ω is solving domain, H_x is tangential component of magnetic field intensity, I_{m} is magnitude of current and b is width of open slot. And the analytical solution of current density (J) is shown as follows:

$$J = -\mathrm{j}\omega\sigma A + J_e = \frac{pI_{\mathrm{m}}}{b\mathrm{sh}ph} \bullet \mathrm{ch}py. \tag{11}$$

Additional, in order to validate the proposed method for eddy current problem with different mediums, Model II is established in this paper (as shown in Fig. 3). For Model II, there is 1 mm width air gap surrounding the conductor, for which the condition number of its system matrix becomes greater than that of Model I, and convergence of solving the equations also becomes worse. Both of two models are under the complex excited current $\boldsymbol{I}_m = (10000 + j0)$ A.

To test the accelerating performance of proposed method on different computation scales, Model I and

Model II have been meshed into three different sizes, shown in Table 1 and Table 2. The mesh of Model II in size B is shown in Fig. 4, and its magnetic field distribution is shown in Fig. 5. Furthermore, the convergence of equations solved using CG and JPCG has been researched. Both of EbE-CG and EbE-JPCG methods are implemented with CPU and GPU separately.

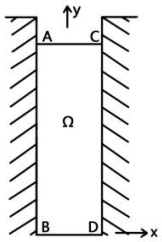


Fig. 2. Current-carrying conductor in an open slot with air surrounded.

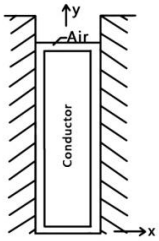


Fig. 3. Current-carrying conductor in an open slot.

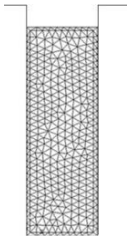


Fig. 4. The mesh of Model II in size B.

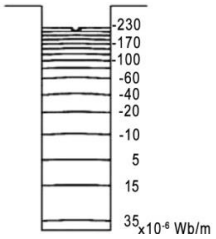


Fig. 5. Magnetic field distribution of Model II.

All the numerical computations are carried out on a server with NVIDIA GTX 660 GPU clocked at 1.0 GHz with 960 cores and 2G DDR5 global memory, and an Intel Xeon E3-1230 CPU 3.3 GHz with 8G global memory. Programming is in C++, and compiled by Visual Studio 2010 and CUDA 5.5.

To reduce the communication cost between CPU and GPU, the whole elements information is transferred to GPU global memory initially. The solving process is operated parallelly on GPU until computation results meet the convergence criterion, then result data is transferred from GPU to CPU. The GPU calculation is fulfilled on different blocks, and the threads on the same block are parallel running. But different block cannot communicate. However, during the CG iteration process, some kinds of steps such as the calculation of $\boldsymbol{r}^{(e)}$ need the information of other relative elements which are not in the same block. To overcome this, if the nodes on the boundary of memory block, the node information is stored on both sides concurrently. A little more memory needed, but high parallelism obtained. For other steps, all the read and write instructions for threads within same warp (a cluster of threads) are operated in the aligned and coalesced way to improve parallel performance.

The calculation results are shown in Table 1 and Table 2. Table 3 is shown the comparison of memory required. Figure 6 is the current density comparison between analytical and numerical solution of Model I. From Fig. 6, we can see that the result calculated using the proposed correlates well with analytical solution, which validates the method.

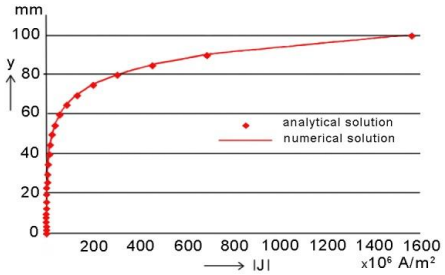


Fig. 6. Comparison of current density with EbE-CG method (Model I).

Table 1: The numerical results of Model I

Mesh Size	Node	Element	Iterations		CPU Time (ms)		GPU Time (ms)	
			CG	JPCG	CG	JPCG	CG	JPCG
A	90	138	56	33	78	62	23	18
B	342	594	100	46	485	359	87	65
C	1080	1953	175	68	1549	1231	239	173

Table 2: The numerical results of Model II

Mesh Size	Node	Element	Iterations		CPU Time (ms)		GPU Time (ms)	
			CG	JPCG	CG	JPCG	CG	JPCG
A	580	683	85	73	2578	1927	753	557
B	905	1511	134	96	6987	5125	1215	843
C	1384	2235	201	137	9768	7254	1441	935

Table 3: Comparison of memory required (Model II)

Mesh Size	Memory Required (kB)		Memory Saved (%)
	EbE	FEM	CG
A	32	79	59.5
B	72	145	50.3
C	107	218	50.9

The distribution of current density in Model II is shown in Fig. 7, which also shows that accurate results can be obtained using EbE-JPCG to eddy current problem with different medium. From the results shown in Table 1 and Table 2, we can see that the convergence of equations solving using JPCG is better than that using CG. For the same model, the GPU processor is faster than CPU due to its high parallelism.

Figure 8 shows three different mesh size level's speed up rate comparison of EbE-CG and EbE-JPCG methods which are fulfilled on GPU for Model II. Figure 9 shows the speed up rate comparison of EbE-JPCG

method fulfilled on GPU for two models.

Both EbE-CG method and EbE-JPCG method are applied to Model II which contains two materials. As shown in Table 2, the time consumed is much more than Model I, however, results indicate overall improved convergence and processing time with increasing mesh size as shown in Figs. 8-9.

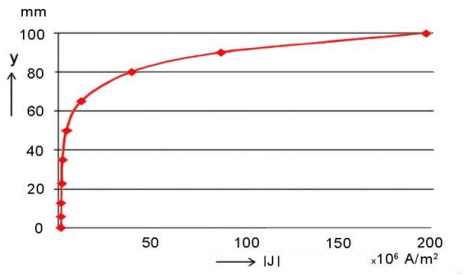


Fig. 7. Distribution of current density for EbE-JPCG method (Model II).

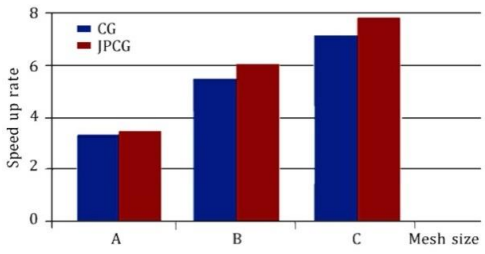


Fig. 8. Comparison of speed up rate for EbE-CG method and EbE-JPCG method.

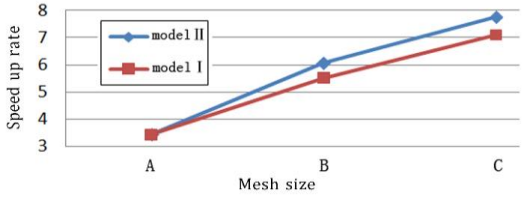


Fig. 9. Comparison of speed up rate for EBE-JPCG method implementation on GPU (Model II).

IV. CONCLUSION

The EbE-JPCG technique and GPU parallel computing platform applied to eddy current problems are the main contributions of this work. This paper presents a comparative analysis of the performance of EbE-CG method and EbE-JPCG method which are fulfilled on CPU and GPU. As shown in Table 1, Table 2 and Fig. 8, EbE-JPCG method converges more quickly than the EbE-CG method. As well, GPU acceleration becomes more effective with increasing mesh size. The numerical results demonstrate that JP method is effective for EbE method and parallel computing. As shown in Table 3, EbE method can save approximately 50% memory space, it is an important contribution for GPU platform which just has a few GB memory. Another contribution of this paper is to provide basis for solving of 3D eddy current problem, as in [8]. The future work currently in progress includes applying the EbE technique and GPU parallel platform to 3D eddy current losses calculation of large power transformer. Considering its serious ill-conditioned, JP method will be ineffective. So a new improved JP method which is also convenient for parallel EbE implementation is included in the ongoing work.

ACKNOWLEDGMENT

This work is supported by National Natural Science Funds (51507105), Science Foundation of Education Department of Liaoning Province, P.R. China (L2013046) and Natural Science Foundation of Liaoning Province, P.R. China (2015020087).

REFERENCES

[1] T. J. R. Hughus, I. Levit, and J. Winget, "An element-by-element solution algorithm for problems of structural and solid mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 36, pp. 241-254, 1983.

[2] A. F. P. Camargos and V. C. Silva, "Performance analysis of multi-GPU implementations of Krylov-subspace methods applied to FEA of electromagnetic phenomena," *IEEE Transactions on Magnetics*, vol. 51, no. 3, March 2015.

[3] T. Okimura, T. Sasayama, and N. Takahashi, "Parallelization of finite element analysis of nonlinear magnetic fields using GPU," *IEEE Transactions on Magnetics*, vol. 49, no. 5, May 2013.

[4] O. Bottauscio, M. Chiampi, J. Hand, et al., "A GPU computational code for eddy-current problems in voxel-based anatomy," *IEEE Transactions on Magnetics*, vol. 51, no. 3, March 2015.

[5] D. M. Fernández, M. M. Dehnavi, and W. J. Gross, "Alternate parallel processing approach for FEM," *IEEE Transactions on Magnetics*, vol. 48, no. 2, pp. 399-402, 2012.

[6] I. Kiss, S. Gyimóthy, Z. Badics, et al., "Parallel realization of the element-by-element FEM technique by CUDA," *IEEE Transactions on Magnetics*, vol. 48, no. 2, pp. 507-510, 2012.

[7] D. Y. Wu, R. Y. Tang, and D. X. Xie, "*Element by element finite element method applied to eddy current problem*," *CEFC'2014*, France, May 25-28, 2014.

[8] D. Y. Wu, X. K. Yan, R. Y. Tang, D. X. Xie, and Z. W. Chen, "GPU acceleration of EBE method for 3-D linear steady eddy current field," *ICEMS'2015*, Thailand, October 25-28, 2015.

[9] D. X. Xie, "Finite element method applied to the calculation of skin effect of current-carrying conductor in an open slot," *HIET Journal*, vol. 4, no. 1, pp. 6-23, 1981. (In Chinese).

GPU-based Electromagnetic Optimization of MIMO Channels

Alfonso Breglia, Amedeo Capozzoli, Claudio Curcio, Salvatore Di Donna, and Angelo Liseno

Università di Napoli Federico II, Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
via Claudio 21, I 80125 Napoli, Italy
a.capozzoli@unina.it

Abstract ─ Strategies to accelerate MIMO channel capacity optimization on GPUs are outlined. The optimization scheme is dealt with by properly facing the main computational issues. In particular, the propagation environment is described by ultrafast Geometrical Optics (GO), singular values are computed by a very fast scheme and the optimizer is a parallel version of the differential evolutionary algorithm. The unknowns are given proper representations to reduce the number of optimization parameters.

Index Terms ─ CUDA, differential evolutionary, Geometrical Optics, GPU, MIMO channel, optimization, singular values.

I. INTRODUCTION

Multiple Input - Multiple Output (MIMO) [1, 2] is a wireless communication technology using multiple antennas in both transmission and reception modalities to increase the channel capacity (CC) over that of a conventional SISO (Single Input - Single Output). CC is a crucial parameter to assess the performance of a communication system, and the problem arises of how defining the most convenient configuration of transmitting (TX) and receiving (RX) antennas to maximize it for a given SNR (Signal to Noise Ratio) and propagation environment.

Different approaches have been proposed to optimize the MIMO CC, see [3, 4] for two representative examples. As it appears from [3, 4], besides signal processing factors (e.g., modulation), two critical aspects emerge when optimizing the performance of a MIMO channel: one is the electromagnetic environment, since the electromagnetic propagation influences the properties of the channel matrix, and the other is the optimization scheme itself. Since both aspects pose a significant computational question, the issue thus arises of how computationally addressing the problem of optimizing a MIMO channel, by firstly determining the most convenient computational resources and algorithms to be exploited and that could make the challenge feasible. Then, how much a MIMO channel can be improved, how the CC depends on the accuracy of the employed electromagnetic model and what can be obtained by a modeling grasping only the essential aspects of the problem should be pointed out, giving general guidelines at the design stage. These points have been up to now overlooked throughout the literature. Our purpose is facing the first point, namely, how much accelerated analysis and optimization can make the goal viable. This entails understanding how to push the performance of both, the employed algorithms and computational resources. Several computational key points should be then considered, since each performance can degrade the problem to unfeasibility:

1. Properly choosing and accelerating a global optimizer since a local optimizer is typically not enough to find the best solutions;
2. Properly choosing and accelerating the approach to compute the MIMO channel matrix;
3. Being the CC related to the singular values (SVs) of the channel matrix, accelerating their computation depending on the problem size (conventional - 4x4, 6x6 - MIMO vs. massive MIMO [5]);
4. Properly representing the unknowns, to manage only the essential optimization parameters;
5. Properly exploiting massively parallel computing platforms as Graphics Processing Units (GPUs).

The paper is organized as follows. The MIMO CC is briefly recalled in Section II, just to provide a formal introduction. Section III addresses points 1) and 4), Section IV point 2) and Section V points 3) and 5). Finally, Sections VI and VII present numerical results and conclusions, respectively.

II. CHANNEL MODEL

Let us consider a narrowband, flat-fading channel, whose CC depends on the distribution of the SVs of the channel scattering matrix [1, 2]. Indeed, given N_{TX} transmitting and N_{RX} receiving antennas embedded in a complex 3D deterministic electromagnetic scenario, the MIMO channel can be described by its complex, N_{TX} x N_{RX} matrix $\underline{\underline{H}}$ [1]. The generic element h_{ij} of $\underline{\underline{H}}$ can be expressed as:

Submitted On: December 29, 2015
Accepted On: March 30, 2016

$$h_{ij} = h(\underline{r}_i^{TX}, \underline{r}_j^{RX}) = \sum_{m=1}^{M} G_m(\underline{r}_i^{TX}, \underline{r}_j^{RX}), \quad (1)$$

where $\underline{r}_i^{TX}$ represents the position of the *i*-th transmitting antenna, $\underline{r}_j^{RX}$ represents the position of the *j*-th receiving antenna, $M(i, j)$ is the number of relevant multi-paths between $\underline{r}_i^{TX}$ and $\underline{r}_j^{RX}$ and $G_m(\underline{r}_i^{TX}, \underline{r}_j^{RX})$ is proportional to the voltage induced on the *j*-th receiving antenna by the *m*-th multipath originated at the *i*-th transmitting antenna.

Under the hypothesis of narrowband, flat-fading channel, AWGN noise at the receivers and equally distributed power among the transmitters [1], the Shannon CC, say *C*, expressed in bit/s/Hz can be calculated by first normalizing $\underline{\underline{H}}$ to its Frobenius norm as [6]:

$$\underline{\underline{\tilde{H}}} = \underline{\underline{H}} \Big/ \sqrt{\frac{\sum_{i=1}^{N_{TX}} \sum_{j=1}^{N_{RX}} |h_{ij}|^2}{N_{TX} N_{RX}}}, \quad (2)$$

and then computing,

$$C = \sum_{k=1}^{r} \log_2 \left(1 + \frac{SNR \sigma_k^2}{N_{TX}} \right), \quad (3)$$

where *r* is the rank of $\underline{\underline{\tilde{H}}}$ and σ_k^2 is its *k*-th SV.

The approach is illustrated in Fig. 1, where the flow-chart boxes correspond to the titles of the Sections III-V.

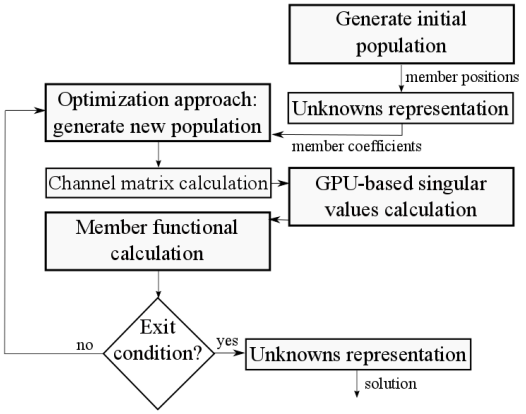


Fig. 1. Flow chart of the approach.

III. THE OPTIMIZATION APPROACH

To enable a satisfactory exploration of the objective functional landscape, the global optimization approach should be chosen to exhibit good convergence properties and to profit of the massive parallelization. In this sense, a less "complex", but massively parallelizable algorithm should be preferred to a more "involved", but "more sequential" scheme. The "genetic-like" differential evolutionary approach has been then chosen as the global optimization scheme due to its main features matching both the above mentioned requirements [7].

The approach exploits a population of N_p members, each member being represented by an array of *D* values, where *D* coincides with the number of optimization unknowns. At the *g*-th iteration (generation), the *k*-th member of the population is denoted by the *D*-dimensional array $\underline{p}_{k\,g} = (p_{k\,g}^{(1)}, p_{k\,g}^{(2)}, \ldots, p_{k\,g}^{(D)})$. The initial population is randomly generated, accounting for some physical constraints enforced by the problem. Starting from the initial population, the algorithm generates a new one by first defining new arrays as (mutation):

$$\underline{m}_{k\,g+1} = p_{k_1\,g} + F(\underline{p}_{k_2\,g} - \underline{p}_{k_3\,g}), \quad (4)$$

where k_1, k_2 and k_3 belong to {1, 2, …, N_p} and three indices mutually different and different also from *k*, and $F \in [0,2]$ is a user defined real and constant factor representing a scale factor of the differential variation $\underline{p}_{k_2\,g} - \underline{p}_{k_3\,g}$.

Following mutation, new trial arrays $\underline{t}_{k\,g+1} = (t_{k\,g+1}^{(1)}, t_{k\,g+1}^{(2)}, \ldots, t_{k\,g+1}^{(D)})$ are generated as follows (crossover):

$$t_{k\,g+1}^{(i)} = \begin{cases} m_{k\,g+1}^{(i)}, & if\ rand(j) \le CR\ or\ j = randD(k) \\ p_{k\,g}^{(i)}, & if\ rand(j) > CR\ or\ j \ne randD(k) \end{cases}, \quad (5)$$

where $rand(j) \in [0,1]$, j=1,2,…,*D*, is the *j*-th evaluation of a random uniform generator, $CR \in [0,1]$ is a user defined crossover constant and $randD(k) \in \{1,2,\ldots,D\}$ is a randomly chosen integer to ensure that the trial array contains at least a mutated element. Following the crossover, the cost values of $\underline{p}_{k\,g}$ and $\underline{t}_{k\,g+1}$ are compared and the one with the largest cost becomes the new population member at generation g+1 (selection).

The operations involved in mutation, crossover and selection and the random number generations are inherently parallel. An issue of the crossover stage is the "random" global memory access, so that particular care has been given to improve memory coalescence.

Unknowns representation

The problem concerns the optimization of the TX and RX antenna locations to maximize the MIMO CC. To profit from a reduction of the number of unknowns, both the TX and RX antennas are assumed to be located on lines and their positions are indirectly searched for by representing them by Legendre polynomials [8] as:

$$x_n = \sum_{k=0}^{K-1} c_k \psi_k(\xi_n) \cdot \quad (6)$$

In Eq. (6), x_n is the generic antenna coordinate on the optimization line, *K* is the number of polynomials, ψ_k is the *k*-th Legendre polynomial, the ξ_n's are uniformly spaced points in [-1,1] and the c_k's become the actual unknowns to be sought for. If *K* is less than the involved antennas, the number of problem parameters is reduced.

Notice that the representation in Eq. (6) is also amenable to enforcement of constraints on minimum and maximum antenna spacings [9].

IV. CHANNEL MATRIX CALCULATION

The method exploited to calculate $\underline{\underline{\tilde{H}}}$ should trade off computational accuracy and speed to execute in an iterative optimization. Calculating at each generation the matrix $\underline{\underline{\tilde{H}}}$ for a large number of antenna configurations by a full wave method would be unfeasible, especially for large scenarios. Opposite to that, Geometrical Optics (GO) is very appealing to quickly provide an approximate solution to Maxwell's equations.

Nevertheless, for electrically large scenarios GO must be properly accelerated by adequate algorithmic structures capable to properly handle the intersections of the rays with the scene objects. Indeed, ray tracing involves two main steps: the search for the intersection between a ray and the geometric primitives (e.g., triangles), and the electromagnetic field transport. The first step can be definitely the most time consuming one, if not properly managed. A brute force approach would be indeed unfeasible due to the large number of intersection tests to be performed. Fortunately, the problem can be faced by tree-like structures which, if properly setup, managed and explored, can significantly reduce the computational complexity. Data structures like KD-tree and BVH (Bounding Volume Hierarchy) [10] can be effectively applied to this purpose and may profit from a high degree of parallelization. Here, the Split BVH (SBVH) scheme set up in [10] has been exploited.

V. GPU-BASED SVs CALCULATION

Computing the SVs of small or large matrices should be dealt with different approaches. Accordingly, the computational scheme to be employed differs if considering conventional or massive MIMOs. In this paper, we address the former case. Furthermore, the number of involved matrices is related to the number of population members of the differential optimizer. Then, at each generation, the SVs of a large number of small sized matrices have to be computed. This task can be efficiently and effectively performed on a GPU as in [11].

The problem of computing the SVs can be recast to the computation of the SVs of a real-valued matrix $\underline{\underline{A}}$. To this end, the approach in [7] consists of three steps. The first step amounts at reducing $\underline{\underline{A}}$ to a bidiagonal matrix, say $\underline{\underline{B}}$, as:

$$\underline{\underline{A}} = \underline{\underline{P}}\,\underline{\underline{B}}\,\underline{\underline{Q}}^T, \tag{7}$$

where $\underline{\underline{B}}$ is a N_{TX} x N_{RX} upper bidiagonal matrix, and $\underline{\underline{P}}$ and $\underline{\underline{Q}}$ are N_{TX} x N_{TX} and N_{RX} x N_{RX} orthogonal Householder matrices, respectively. The bidiagonalization step consist of applying a sequence of Householder transformations [12] to the matrix $\underline{\underline{A}}$, which zero the elements below the diagonal and to the right of the first superdiagonal. In the second step, $\underline{\underline{B}}$ is transformed to a tridiagonal matrix $\underline{\underline{T}} = \underline{\underline{B}}^T\,\underline{\underline{B}}$. Finally, in the third step, the symmetric tridiagonal eigenvalue problem is solved by a bisection method based on the use of Sturm sequences by restricting the search range using the Gershgoring circle theorem [11].

The motivation for computing the tridiagonal matrix $\underline{\underline{T}}$ as above is due to the fact that the explicit formation of $\underline{\underline{T}}$ should be avoided for numerical reasons since it may introduce non-negligible relative errors, especially in the computation of the smallest SVs [12]. However, the exploited approach is meant for those applications, as the one at hand, in which the smallest SVs have very low relative weight and may be considered irrelevant.

In summary, the problem of computing the SVs is recast as a "guided" bisection, which is amenable to parallelization.

VI. NUMERICAL RESULTS

The optimizer, the ray tracer and the SVs calculation have been implemented in parallel GPU (CUDA) and multi-core CPU (C++ with OpenMP directives) languages. For the CPU case, the SVs have been achieved using the third party Eigen library.

We consider a circular cylinder with radius 10λ centered at the origin of the *Oxyz* reference system and a plate of width 50λ, parallel to the *yz* plane and located at x=30λ. The cylinder and the plate are perfectly conducting and have a height of 15λ. The scene has been discretized with 95458 triangles. This example points out how much computation time can be saved by the approach and provides an answer to the points raised in Section I and a perspective to design tools.

The optimizer can position an arbitrary number of transmitting and receiving antennas on lines with arbitrary spatial orientations. Here, N_{TX} = 4 and N_{RX} = 4. The TX and RX antennas have been located on lines lying on the *xy* plane, parallel to the *x*-axis and passing by (15λ, -40λ, 0λ) for the TX and by (15λ, 40λ, 0λ) for the RX antenna. The antenna positions have been represented using K=3 and minimum and maximum spacing of λ/4 and 2λ, respectively, have been enforced to control the maximum array size and mutual coupling. The SNR has been fixed to 20 dB.

For computational convenience, the optimization has been run with a population of 1000 elements, grouped in 10 subgroups including those configurations sharing the same positions of the TX antennas and different positions of the RX ones. The optimization has been run for a number of 50 generations, with CR=0.4 and F=0.7.

The code has been run on a workstation equipped with two Intel Xeon E5-2650 2.00GHz, Eight core processors each and four NVIDIA Kepler K20c cards, but with multi-GPU disabled. Figure 2 displays an OpenGL rendering of the MIMO channel with the optimized antenna locations. The figure also depicts the GO rays connecting the TX antennas (the pink dots) with the RX ones (not appearing in the image). As it can be seen, multiple interactions have been accounted for as well as diffraction from the plate border. Diffraction from the plate corners and the cylinder ends have been neglected for simplicity. The optimized antenna positions are reported in Table 1. As it can be seen, the TX and RX antennas occupy almost symmetric locations due to the problem symmetry. The GPU code has run in about 4.5 hours, gaining a speedup of about 5 as compared to the CPU execution obtained by running 32 CPU threads. The optimized channel capacity has been 22.3 bps/Hz, a value which well agrees with the statistical distribution of channel capacities for random channels with 4 transmitting and 4 receiving antennas reported in [2, Fig. 7].

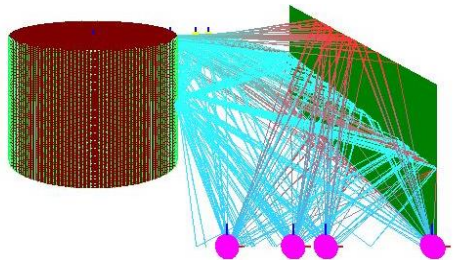


Fig. 2. OpenGL rendering of the MIMO channel with optimized antenna locations.

Table 1: Optimized TX and RX antenna positions

Antenna	*x*-coord.	Antenna	*x*-coord.
TX 1	7.0λ	RX 1	7.08λ
TX 2	12.6λ	RX 2	12.9λ
TX 3	18.0λ	RX 3	17.1λ
TX 4	20.9λ	RX 4	22.0λ

VII. CONCLUSIONS

A GPU-based approach to accelerate MIMO CC optimization has been presented using ultrafast Geometrical Optics (GO), a very fast SV calculation scheme and a parallel version of the differential evolutionary algorithm. A speedup of 5 has been achieved against a multi-core CPU implementation.

ACKNOWLEDGMENT

Work partially funded by the Italian Ministry of Education, University and Research (MIUR), project PON01_02425 “Services for wIreless netwoRk Infrastructure beyOnd 3G” (SIRIO).

REFERENCES

[1] G. J. Foschini and M. J. Gans, “On limits of wireless communications in a fading environment when using multiple antennas,” *Wireless Personal Commun.*, vol. 6, no. 3, pp. 311-335, Mar. 1998.

[2] J. Bach Andersen, “Array gain and capacity for known random channels with multiple element arrays at both ends,” *IEEE J. Selected Areas Commun.*, vol. 18, no. 11, pp. 2172-2178, Nov. 2000.

[3] U. Olgun, et al., “Optimization of linear wire antenna arrays to increase MIMO channel using swarm intelligence,” *Proc. of the 2nd Europ. Conf. on Antennas Prop.*, Edinburgh, UK, pp. 1-6, Nov. 11-16, 2007.

[4] M. A. Mangoud, “Optimization of channel capacity for indoor MIMO systems using genetic algorithm,” *Progr. Electromagn. Res. C*, vol. 7, pp. 137-150, 2009.

[5] E. G. Larsson, et al., “Massive MIMO for next generation wireless systems,” *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 186-195, Feb. 2014.

[6] N. Noori and H. Oraizi, “Evaluation of MIMO channel capacity in indoor environments using vector parabolic equation method,” *Progr. Electromagn. Res. B*, vol. 4, pp. 13-25, 2008.

[7] R. Storn and K. Price, “Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces,” *J. Global Opt.*, vol. 11, no. 4, pp. 341-359, Dec. 1997.

[8] A. Capozzoli, et al., “Field sampling and field reconstruction: a new perspective,” *Radio Sci.*, vol. 45, RS6004, pp. 31, 2010, doi: 10.1029/ 2009RS004298.

[9] A. Capozzoli, et al., “FFT & aperiodic arrays with phase-only control and constraints due to super-directivity, mutual coupling and overall size,” *Proc. of the 30th ESA Antenna Workshop on Antennas for Earth Observ., Science, Telecomm. and Navig. Space Missions*, Noordwijk, The Netherlands, May 27-30, 2008, CD ROM.

[10] A. Breglia, et al., “Comparison of acceleration data structures for electromagnetic ray tracing purposes on GPUs,” *IEEE Antennas Prop. Mag.*, vol. 57, no. 5, pp. 159-176, Oct. 2015.

[11] A. Capozzoli, et al., “Massive computation of singular values of small matrices on GPUs,” *Proc. of the Int. Workshop on Comput. Electromagn.*, Izmir, Turkey, pp. 36-37, July 1-4, 2015.

[12] G. H. Golub and C. Reinsch, “Singular values decomposition and least squares solutions,” *Numer. Math.*, vol. 14, pp. 403-420, 1970.

Fast and Parallel Computational Techniques Applied to Numerical Modeling of RFX-mod Fusion Device

Domenico Abate [1,2], Bruno Carpentieri [3], Andrea G. Chiariello [4], Giuseppe Marchiori [2], Nicolò Marconato [2], Stefano Mastrostefano [1], Guglielmo Rubinacci [5], Salvatore Ventre [1], and Fabio Villone [1]

[1] DIEI, Università di Cassino e del Lazio Meridionale, Loc. Folcara, 03043 Cassino (FR), Italy
s.mastrostefano@unicas.it, ventre@unicas.it, villone@unicas.it

[2] Consorzio RFX, Corso Stati Uniti 4, Padova, Italy
domenico.abate@igi.cnr.it, giuseppe.marchiori@igi.cnr.it, nicolo.marconato@igi.cnr.it

[3] Nottingham Trent University, School of Science and Technology, Burton Street, Nottingham NG1 4BU, UK
bruno.carpentieri@ntu.ac.uk

[4] DIII, Seconda Università di Napoli, Via Roma 29, Aversa (CE), Italy
andreagaetano.chiariello@unina2.it

[5] DIETI, Università di Napoli Federico II, Via Claudio 21, 80125, Napoli
rubinacci@unina.it

Abstract ― This paper presents fast computational techniques applied to modelling the RFX-mod fusion device. An integral equation model is derived for the current distribution on the active coils of the conducting structures, and the input-output transfer functions are computed. Speed-up factors of about 200 can be obtained on hybrid CPU-GPU parallelization against uniprocessor computation.

Index Terms ― Fusion plasma devices, GPUs, HPC, integral formulation, parallelism.

I. INTRODUCTION

Modelling fusion devices is computationally very challenging due to the electromagnetic interaction of the fusion plasma and the surrounding conducting structures, which makes the problem inherently multiphysics. The evolution of the plasma may exhibit unstable modes, thus exacerbating the aforementioned problems and requiring a feedback controller. The design of such control system requires rather accurate response model of the overall system plasma plus conductors. Therefore, fast parallel techniques are often required to make the computations affordable [1, 2]. In this paper, we analyze the RFX-mod device [3], a medium size (major radius R = 2 m, minor radius a = 0.46 m) toroidal device particularly suited to explore innovative concepts in plasma control. Passive and active conductors are very important to determine the overall properties and performances of such feedback system and therefore they should also be adequately represented in any realistic model. The main conducting structures are the vessel (needed to have the vacuum inside the machine), the shells (highly conducting sheets needed for passive stabilization), the mechanical structure, hosting the active control coils. Figure 1 shows the 3D hexahedral mesh used.

In particular, RFX-mod is equipped with a state-of-the-art control system made by 192 (4 poloidal x 48 toroidal) independently fed active coils (Fig. 1), with more than 600 magnetic sensors acquired in real time. This makes RFX-mod on the one hand very challenging for numerical modelling but on the other hand an ideal test-bed for validating the predicting capabilities of computational tools. We compute the input-output transfer functions of the system, assuming as input the currents or the voltages of the active coils and as output suitable magnetic measures [4]. The presence of an axisymmetric plasma evolving through equilibrium states is self-consistently taken into account [1].

The computer solution of such a problem is very expensive, due to the complexity of the 3D geometry and the plasma contribution. The use of High Performance Computing (HPC) cluster is mandatory. The GPU architecture has a large amount of cores designed to run a large number of execution threads at the same time; the computational model used is the single instruction,

Submitted On: December 31, 2015
Accepted On: May 16, 2016

multiple data (SIMD), where concurrent threads execute the same code (called Kernel) on different data. In the present work, we focus our attention on a hybrid multi-node system for modeling RFX-mod devices.

The paper is organized as follows. Section II describes the model, while in Section III we illustrate the computational technique. Section IV reports the results and draws the conclusions.

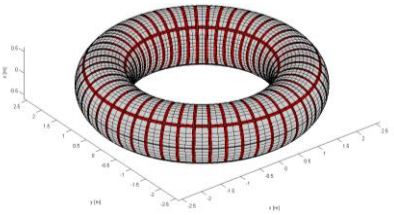


Fig. 1. Mesh used for the analysis of the problem.

II. MODEL

We consider a system of 3D conductors V_c discretized with a finite elements mesh. We use an integral formulation, which assumes as primary unknown the current density in V_c. We introduce the electric vector potential **T**, such that $\mathbf{J} = \nabla \times \mathbf{T}$, and then we expanded **T** in terms of edge elements $\mathbf{N}_k$, we have:

$$\mathbf{J} = \sum_k I_k \nabla \times \mathbf{N}_k \,. \tag{1}$$

Imposing Ohm's law in weak form, we get [1,2,8]:

$$\underline{\underline{L}} \frac{d\underline{I}}{dt} + \underline{\underline{R}}\,\underline{I} + \frac{d\underline{U}}{dt} = \underline{V}, \tag{2}$$

$$L_{i,j} = \frac{\mu_0}{4\pi} \int_{V_c} \int_{V_c} \frac{\nabla \times \mathbf{N}_i(\mathbf{r}) \cdot \nabla \times \mathbf{N}_j(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} dV\, dV', \tag{3}$$

$$R_{i,j} = \int_{V_c} \nabla \times \mathbf{N}_i \cdot \boldsymbol{\eta} \cdot \nabla \times \mathbf{N}_j dV \,. \tag{4}$$

In these equations, $\underline{I}$ is the vector of degrees of freedom I_k in (1), $\underline{V}$ is the vector of externally applied voltages and $\boldsymbol{\eta}$ is the resistivity tensor. Matrix $\underline{\underline{L}}$ is a fully populated square matrix, which is the 3D analogue of mutual inductance of a system of magnetically coupled conductors; conversely, $\underline{\underline{R}}$ matrix is sparse and represents the resistance matrix of the 3D conductors. The quantity $\underline{U}$ is the magnetic flux due to plasma currents [1, 8]:

$$\underline{U} = \underline{\underline{M}}\,\underline{j}_S\,,\ \underline{j}_S = \underline{\underline{S}}\,\underline{\hat{\psi}}_e\,,\ \underline{\hat{\psi}}_e = \underline{\underline{Q}}\,\underline{I}\,, \tag{5}$$

where $\underline{j}_S$ are equivalent currents located on a coupling surface, $\underline{\underline{M}}$ is a mutual inductance matrix between the equivalent current and the 3D conducting structures, $\underline{\hat{\psi}}_e$ is the external magnetic flux, $\underline{\underline{Q}}$ is a matrix representing Biot-Savart integral [1] and $\underline{\underline{S}}$ is the plasma response matrix [8].

Combining (2)-(5), finally we get [8]:

$$\underline{\underline{L}}^* \frac{d\underline{I}}{dt} + \underline{\underline{R}}\,\underline{I} = \underline{V}\,, \quad \underline{\underline{L}}^* = \underline{\underline{L}} + \underline{\underline{M}}\,\underline{\underline{S}}\,\underline{\underline{Q}}, \tag{6}$$

to which we can add the expression for the magnetic field and flux perturbations $\underline{y}$ at given points, linearly related to 3D currents through a suitable matrix $\underline{\underline{C}}$ [1,8]:

$$\underline{y} = \underline{\underline{C}}\,\underline{I}\,. \tag{7}$$

Equations (6)-(7) represent the model; they can be easily recast in standard state space form. In the present paper, they are used to get the frequency-domain transfer functions between the inputs (voltages or currents in active coils) and the outputs (linear combinations of magnetic measurements). In doing so, the inversion of a complex matrix is required. Indeed, we split the unknowns into three subsets; the corresponding subset of indices of the various matrices are identified with the following suffix: "p" (passive structures), "m" (measurement coils), "a" (active coils), so that Equation (6) reads as:

$$\begin{aligned} &\left(j\omega \underline{\underline{L}}^*_{pp} + \underline{\underline{R}}_{pp} \right) \underline{I}_p + j\omega \underline{\underline{L}}^*_{pa} \underline{I}_a = 0, \\ &\underline{\underline{L}}^*_{mp} \underline{I}_p + \underline{\underline{L}}^*_{ma} \underline{I}_a = \underline{\Phi}_m, \end{aligned} \tag{8}$$

where $\underline{\Phi}_m$ represent the fluxes induced at measurements coils (i.e., the output of the system). After some simple algebraic manipulations it turns out:

$$\begin{aligned} &\underline{I}_p = -j\omega \left(j\omega \underline{\underline{L}}^*_{pp} + \underline{\underline{R}}_{pp} \right)^{-1} \underline{\underline{L}}^*_{pa} \underline{I}_a = \underline{\underline{H}}(j\omega) \underline{I}_a, \\ &\underline{\Phi}_m = (\underline{\underline{L}}^*_{mp} \underline{H}(j\omega) + \underline{\underline{L}}^*_{ma}) \underline{I}_a = \underline{\underline{T}}(j\omega) \underline{I}_a. \end{aligned} \tag{9}$$

Equation (9) can be used to evaluate Φ_m for a given assigned unitary current flowing in excitation coil. This transfer function can be used to design the feedback control. For each frequency and each active coil, we set Ia to 1 (the known terms in the Eq. (9)) and find the Flux for all measurement coils (unknowns variables).

III. FAST TECHNIQUES

In order to speed up the overall computation we move in two directions:

- parallelize the matrix assembly phase;
- accelerate the inversion of system (9).

A. Parallel assembly strategies

Matrices $\underline{\underline{L}}$ and $\underline{\underline{Q}}$ are very expensive to assemble. For $\underline{\underline{L}}$ matrix, parallelization can be achieved grouping elements of nodes into boxes, distributing boxes among processors, and performing the element-element integration independently on each processor. The locally assembled matrix is then compressed (see [2]).

The computation of matrix $\underline{\underline{Q}}$ is the most time consuming part of the assembly algorithm. In order to reduce this cost, in the present work, parallel assembly is implemented on multi CPUs and multi GPUs environment. Here we take advantage of the fact that

Biot-Savart integral computation for elements and field points, are independent from each other. Of course, different implementations are necessary to adapt the parallel computation to the two different hardware architectures. In the following, we briefly recall the main features of the two algorithms.

In multi CPUs environment we propose a standard parallel strategy using a simple domain decomposition approach that distributes the field points equally among the processors. After the local computations, a reduction operation is required to retrieve the complete matrix from each MPI process. This strategy scales linearly with the number of the processors.

In multi GPUs environment we propose to assign to each computational thread the evaluation of a contribution of the Biot-Savart integral corresponding to a given element and a given field point. All the contributions are summed on the CPU. The algorithm is briefly summarized in the follow, see [1] for details:

1) Allocate temporary data for storing the local contribute (CPU).
2) Compute the considered element and source point from the thread and block index (GPU).
3) Compute the shape function related to the considered element (GPU).
4) Compute the local contribution (GPU).
5) Return the partial matrix to the host memory (CPU).
6) Scatter the output data on the complete matrix on the host (CPU).

The final step is due to uncoalescent memory access needed to store the results in the final matrix and possible race conditions when two different contributions are summed in the same location. The dimension of the matrix can be huge compared to the on board GPU memory (which is typically of a few GB). Step 5 involves memory transfer from GPU to Host memory, but fortunately this has no impact on the overall performance of the code. We point to [9] for more sophisticated approaches not considered here.

B. Speed up of the linear system inversion

As far as the inversion of the linear system involved in (9) is concerned, it is worth noting that $\underline{\underline{L}}^{*}_{xy}$ are fully populated submatrices of matrix $\underline{\underline{L}}^{*}$ and $\underline{\underline{R}}_{pp}$ is a sparse positive definite matrix. Using a direct solver, the cost of the inversion procedure is $O(N^3)$, N being the number of unknowns present in the passive part of the device. When geometric details are added and/or a great accuracy is required in the computation, it is easy to exceed quickly the computational resources available on a uniprocessor system. The use of powerful computing facilities can help in the search of additional speed and increase the size of the solvable problems [5].

Nevertheless, there are cases in which parallelization fails poorly. For this problem, an approximated compression technique is mandatory. The authors successfully applied these methods for the study of plasma fusion devices [2] as well as in other fields (e.g., NDT [6]). These techniques are based on an effective low-rank approximation of the submatrix representing the far interaction between well separated parts of the device. The matrix-by-vector product $\underline{\underline{L}}^{*}_{ij} I_j$ related to these parts is replaced by an accurate low cost operator (the complexity is asymptotically only $O(N)$). Finally, the inversion in (9) can be performed by an iterative method (such as the GMRES method). It is worth noting that the preconditioner (essential for any iterative solver) is $\underline{\underline{R}}^{-1}_{pp}$, which can be computed in fast and accurate way by the means of Cholesky decomposition. It is important to stress that its factorization and back substitution is very cheap using a single CPU. Moreover the preconditioner turns out to be very effective, being the number of iterations required to converge very small.

IV. RESULTS

The computational cluster used for the evaluation of the numerical performances is made by two nodes. Each node consist of 16x cores Intel Xeon CPU E5-2690 (@ 2.90 GHz processor, 20 MB L2), 128 GB RAM, 2×NVIDIA Kepler K20 (2496 cores, 6 GB VRAM).

A. 2D validation and transfer function computation

First of all, a numerical validation of the procedure is carried out. We generated a 3D mesh which fictitiously reproduce an axisymmetric geometry, so that a 2D code (CREATE_L [7]) can be used as benchmark. We computed the transfer function $\underline{\underline{T}}$ defined in the previous section with the two codes, finding a very good agreement, as shown in Fig. 2. This confirms the correctness of the procedure.

In order to show the actual effect due to the presence of the plasma, we compare the results obtained with and without plasma on the full 3D mesh described above. The plasmaless computation is in fact a purely magneto-quasi-static calculation. The number of elements of the mesh is equal to 30907, the number of nodes is 81550. The number of unknowns in the passive structure (i.e., the dimension of the matrix to invert) is 22619. The results are reported in Fig. 3. Evidently, the presence of the plasma has an effect not only on the dynamical properties of the model (e.g., the phase behavior at high frequencies), but also on the static gain (amplitude at zero frequency limit). This is not surprising, since the plasma affects also the magnetostatic coupling between active coils and sensors, because it reacts to external static magnetic field perturbations, so as to reach a different equilibrium configuration and hence, modifying the whole magnetic field map in the surrounding regions.

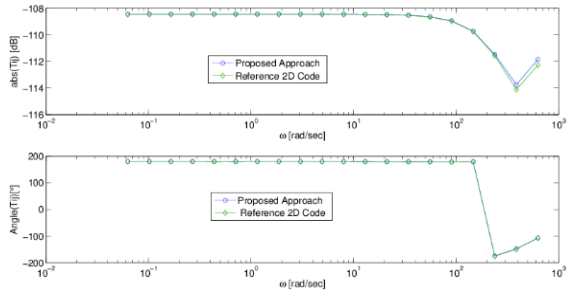


Fig. 2. Comparison of one element of the transfer function $\underline{T}$: proposed approach and reference 2D code.

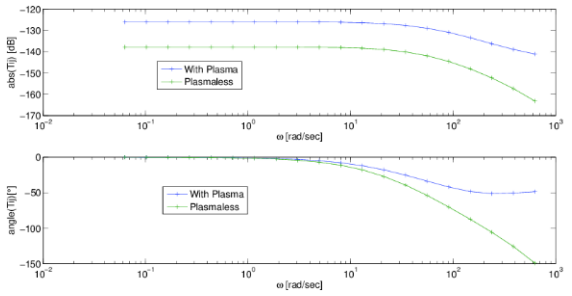


Fig. 3. Effect of plasma on the transfer function.

B. Numerical issues

Regarding the speedup of the matrix assembly, using 25 cores the time required to compute the compressed matrix $\underline{L}$ is about 90 s, the total time required to compute the plasma matrices is about 549 s (540 s of this time is due to the computation of $\underline{Q}$ matrix). In Fig. 4 we report the speedup for assembling $\underline{Q}$ matrix, defined as the assembly time required by one CPU divided by the time obtained using a parallel multi GPUs. Using standard parallel strategy (multi CPUs) the maximum achievable speed up on the proposed computational system is limited to 32.

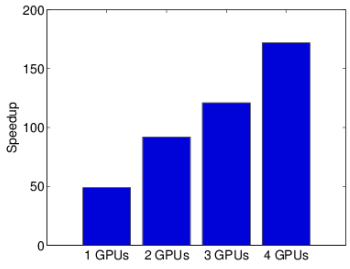


Fig. 4. Speedup for Q-matrix assembly

The time required for each single inversion is about 17.5 s. The total time for all inversions is about 7000s.

The number of iterations required by GMRES to converge increases with the excitation frequency. Without the plasma (i.e., the response due to only the passive structures) the number of iterations required by GMRES to converge is 21 at a frequency of 100 Hz and 9 at 10 Hz. If the plasma is present the number of iteration is 41 at frequency of 100 Hz and 9 at 10 Hz. This is coherent with the general expectation that the used preconditioner is more effective at lower frequencies and without plasma.

V. CONCLUSIONS

We have presented fast parallel techniques for the computation of input-output transfer functions on the RFX-mod fusion devices on hybrid architectures, featuring multiple CPUs and GPUs. The peculiarities of fusion devices make this approach particularly effective in significantly improving the performances of the computation, allowing speed-ups up to almost 200 with respect to standard computations.

REFERENCES

[1] F. Villone, A. G. Chiariello, S. Mastrostefano, A. Pironti, and S. Ventre, "GPU-accelerated analysis of vertical instabilities in ITER including three-dimensional volumetric conducting structures," *Plasma Phys. Control. Fusion*, vol. 54, no. 8, 2012.

[2] G. Rubinacci, S. Ventre, F. Villone, and Y. Liu, "A fast technique applied to the analysis of resistive wall modes with 3D conducting structures," *Journal of Comp. Phys.*, vol. 228, no. 5, pp. 1562-1572, 2009.

[3] P. Sonato, et al., *Fusion Eng. Des*., vol. 66-68, pp. 161, 2003.

[4] F. Villone, et al., "ITER passive and active RWM analysis with the CarMa code," *38th EPS Conference*, paper P5.107, 2011.

[5] R. Fresa, G. Rubinacci, and S. Ventre, "An eddy current integral formulation on parallel computer systems," *Int. Journal for Numerical Methods in Engineering*, vol. 62, no. 9, pp. 1127-1147, 2005.

[6] G. Rubinacci, A. Tamburrino, and S. Ventre, "Fast numerical techniques for electromagnetic non-destructive evaluation," *Nondestr. Testing Eval.*, vol. 24, pp. 165-194, 2009.

[7] R. Albanese and F. Villone, "The linearized CREATE-L plasma response model for the control of current, position and shape in tokamaks," *Nucl. Fusion*, vol. 38, no. 5, pp. 723, 1998.

[8] A. Portone, et al., "Linearly perturbed MHD equilibria and 3D eddy current coupling via the control surface method," *Plasma Phys. Control. Fusion*, 50, 085004, 2008.

[9] A. Capozzoli, et al., "Speeding up aperiodic reflectarray antenna analysis by CUDA dynamic parallelism," *Proc. of the Int. Conf. on Numerical Electromagn. Model. and Opt. for RF, Microwave and Terahertz Appl*., Pavia, Italy, pp. 1-4c, 2014.

Parallel Implementations of Multilevel Fast Multipole Algorithm on Graphical Processing Unit Cluster for Large-scale Electromagnetics Objects

Nghia Tran and Ozlem Kilic

Department of Electrical Engineering and Computer Science
The Catholic University of America, Washington, DC, 20064, USA
16tran@cua.edu, kilic@cua.edu

Abstract ─ This paper investigates solving large-scale electromagnetic scattering problems by using the Multi-level Fast Multipole Algorithm (MLFMA). A parallel implementation for MLFMA is performed on a 12-node Graphics Processing Unit (GPU) cluster that populates NVidia Tesla M2090 GPUs. The details of the implementations and the performance achievements in terms of accuracy, speed up, and scalability are shown and analyzed. The experimental results demonstrate that our MLFMA implementation on GPUs is much faster than (up to 37x) that of the CPU implementation.

Index Terms ─ Graphics Processing Unit (GPU), Multilevel Fast Multipole Algorithm (MLFMA).

I. INTRODUCTION

Over the past twenty years, various numerical techniques have been developed to reduce the computational time and memory requirements of full-wave electromagnetic models without significant loss of accuracy, including adaptive integral method (AIM) [1], impedance matrix localization (IML) [2], fast multipole method (FMM) [3], and multi-level fast multipole algorithm [4]. Compared with the others, MLFMA is among the most suitable techniques for large-scale problems. It reduces the computational complexity of the method of moments (MoM) from $O(N^2)$ to $O(N\log N)$, where N denotes the number of unknowns, whereas AIM, IML and FMM have the complexities of $O(N^{3/2}\log N)$, $O(N^2\log N)$, and $O(N^{3/2})$, respectively.

Recently, many authors have investigated the parallelization of MLFMA on CPU clusters [5] in solving problems of hundreds of thousands to millions of unknowns. In [6], CPU clusters were used to implement MLFMA using Open MP and MPI library to solve a billion unknowns. Multi-GPU implementation was also investigated on a single node, multi-GPU computer without using the MPI library [7]. In this paper, we demonstrate the implementation of MLFMA for electromagnetics problems on GPU clusters by using the MPI library.

We demonstrate the parallelization of MLFMA on a 12-node GPU cluster each of which is populated with an NVidia Tesla M2090 GPU. An MVAPICH2 implementation of MPI is used for cluster parallel programming. This paper is the continuation of our GPU implementation of FMM by using GPU clusters. In [9] and [10], GPU implementation for single level Fast Multipole Method (FMM) solves only the maximum problem size up to 656K unknowns on 13 nodes. In this paper, our MLFMA implementation on GPU cluster can solve up to 1.1 M unknowns. We demonstrate that the implementation of MLFMA on GPUs is faster than that of the CPU. The performance of the implementation is analyzed by using a PEC sphere.

The rest of the paper is organized such that Section II provides an overview of MLFMA. Section III presents the parallel implementation of MLFMA on GPU clusters. Experimental results are discussed in Section IV, followed by the conclusions in Section V.

II. OVERVIEW OF THE MULTILEVEL FAST MULTIPOLE ALGORITHM

The fundamental principles of MLFMA and its applications in electromagnetics have been studied in literature [4]-[5]. In this section, we provide a brief overview to help our discussion on its parallel implementation, which is presented in Section III.

MLFMA was invented based on the grouping concept to accelerate the iterative solution of the linear equation system ZI = V of the Method of Moment (MoM), where I represents the unknown currents, V depends on the incident field, and Z is the impedance matrix. The main idea of the grouping concept is shown in Fig. 1, where the M edges in the mesh of a given structure are categorized into an N-level tree structure connecting groups of different sizes from the finest (level N) to the coarsest level (level 0). Based on the groups' proximity, the impedance matrix Z can be split into two matrices, Z^{near} and Z^{far}, corresponding to near and far interactions as shown in Equation (1):

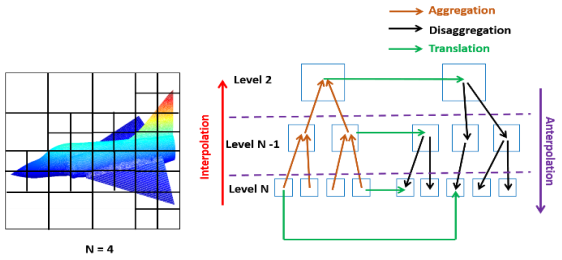
$$\sum_{m'}^{M} Z_{mm'} I_{m'} = \sum_{m'}^{M} Z_{mm'}^{near} I_{m'} + \sum_{m'}^{M} Z_{mm'}^{far} I_{m'} = V_m, \qquad (1)$$

where m and m' are observation and source edges in the

Submitted On: February 27, 2016
Accepted On: May 10, 2016

mesh, respectively.

The Z^{near} matrix comprises of interactions between edges in spatially nearby groups, and is computed and stored using the conventional MoM [8]. During the iterative solution, the near matrix is calculated by the regular sparse matrix-vector multiplications (MVMs). The remaining edges, whose parents are near, constitute the far term as shown in Fig. 1 (b). By treating the interactions between the edges that are spatially far-away using MLFMA, Z^{far} matrix does not need to be explicitly computed and stored. Instead, the far components can benefit from the fast MVMs during the iterative solution. The Z_{far} matrix is factorized into radiation, receive and translation functions, as explained in [4].



(a) (b)

Fig. 1. MLFMA general concepts: (a) concept of the MLFMA tree, and (b) MLFMA concept of far interactions.

The far component is calculated through five main stages: aggregation, translation, and disaggregation, interpolation and anterpolation as shown in Fig. 1 (b).

In the aggregation stage, radiated fields among the groups from level N (the finest level) to level 2 are calculated. At the finest level N, the radiation functions for a group are computed by combining the radiation patterns of the basic function of all edges in this group. From level N-1 to level 2, the radiation functions for each group are computed from the combination of the radiation function of its children group of the finer level using shifting and interpolation.

In the disaggregation stage, the receive functions at each group are computed from level 2 to level N by combining the local incoming waves due to translation and the incoming waves from parent groups of the coarser level using shifting and anterpolation.

The translation stage is identical to FMM [3], and the details of interpolation and anterpolation can be found in [5].

III. PARALLELIZATION OF MLFMA ON GPU CLUSTERS

In this section, we provide an overview of our implementation on GPU. The implementation consists of pre-processing, processing and post-processing. The geometry mesh data resulting from the pre-processing step is transferred to the GPU memory, and the entire computation is performed on the GPU. The user defined results such as radar cross section, scattered fields are post-processed on CPU.

The GPU cluster used for our implementation consists of 12 computing nodes. Each node has a dual 6-core 2.66 GHz Intel Xeon processor, 48GB RAM along with one NVidia Tesla M2090 GPU running at 1.3 GHz supported with 6GB of GPU memory. The nodes are interconnected through the InfiniBand interconnection. The cluster populates CUDA v6.0 and MVAPICH2 v1.8.1 (a well-known implementation of Message Passing Interface (MPI)).

In the processing step, the workload of the computational task is equally distributed among the computing nodes, and the inter-node communication is minimized. This is achieved by uniformly distributing the total number of groups, M, among the n computing nodes. The parallelization of the GPU cluster implementation is performed at two levels: (i) among the computing nodes using MPI library, and (ii) within the GPU per node using CUDA programming model. Within each node, the CUDA thread-block model is utilized to calculate the workload assigned to that node. We only present the far interactions in this paper, since the near field and V vector calculations implementations can be found in [9]-[10].

All CUDA kernels are implemented to calculate Z^{near} matrix, and far interactions which includes the radiation/receive functions, translation matrix, and interpolation/anterpolation matrices. In fast matrix-vector multiplication (MVM), CUDA kernel is also utilized to compute the radiated fields, translation fields and received fields in the aggregation, translation and disaggregation stages, respectively. MPI library is also used to gather results from each node in the end of MVM stage.

A. Far interactions calculations

This task comprises of five calculations: radiation, and receive functions, interpolation, anterpolation and translation matrices.

(i) Radiation and Receive Function Calculations

The first step in the far interaction calculations is the calculation of the radiation, T^E, and receive, R^E, functions for Z^{far} matrix. They are complex conjugates of each other. Thus their implementations are similar. Following the M group distribution, each node handles the calculations of K directions for M_{node} groups. Given this amount of workload per node, the CUDA kernel is launched with $M_{node}.K$ blocks such that each block implements M_{group} radiation/receive function calculations at a given direction, resulting in a total of $M_{node}.K$ blocks per node.

(ii) Translation Matrix Calculation

The second task for far interactions is the calculation of the translation matrix, T_L. The workload for the T_L calculations is also distributed across the nodes following

the group-based technique. By careful investigations, allocating a CUDA block on a single row of the matrix is the efficient way for the translation matrix calculation to save memory requirements. Each CUDA block is assigned to compute one sparse row of the T_L matrix for a given direction, and each thread computes one element in that row.

(iii) Interpolation and Anterpolation Matrices

The third task for the far interactions is the calculation of interpolation and anterpolation matrices. They are transposes of each other. Thus their implementation is similar. Each node handles the calculations of $K_{children/node}$ rows of $K_{children}$xK_{parent} interpolation matrix, where $K_{children}$ is number of directions of a finer level, and K_{parent} is number of directions of a coarser level. The CUDA kernel is launched with $K_{children/node}$ blocks per node. In each block, the maximum number of threads (1024 threads) are utilized in order to implement the full number of K_{parent} directions.

B. Fast matrix-vector multiplication

The next stage for the processing is the solution for the linear system where we employ the iterative method known as the biconjugate gradient stabilized method (BiCGSTAB). The calculation of $Z^{far}I$ comprises of five stages: aggregation, translation, interpolation, anterpolation and disaggregation, as shown in Fig. 2. Using a group-based partitioning technique, the unknown current vector I (N_{edges}x1) is distributed across the computing nodes on GPU clusters.

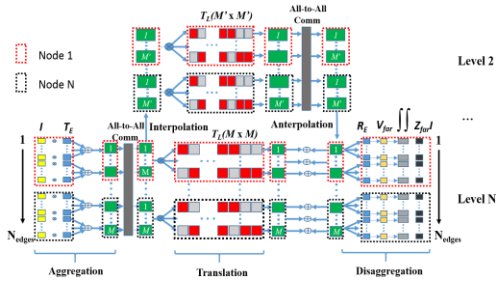


Fig. 2. Far matrix-vector-multiplication in parallel.

In the aggregation stage, at level N, each node computes the radiated fields for M_{node} groups for K directions by multiplying the unknowns I with their corresponding radiation functions, T^E, and accumulating within each group. After the aggregation step, an all-to-all communication is employed by each node to broadcast the radiated fields to all other nodes. The radiated fields from level N-1 to level 2 are computed by multiplying interpolation matrices with radiated fields of children groups at lower levels.

In the translation, the radiated fields at each direction are calculated from the sum of the multiplication of the translation matrix and the radiated fields, and the received fields from parent groups at upper levels using anterpolation.

In the disaggregation stage, the received fields of all M group at level N are multiplied with the corresponding receive functions, and integrated over the partitioned K directions of the unit sphere. The far components of MVM are then incorporated with the near components of MVM. At the end of MVM, the partial results from all nodes are summed together and all nodes are updated.

IV. EXPERIMENTAL RESULTS

A. Accuracy

First, we verify the accuracy of our GPU implementation by calculating the radar cross section (RCS) of a 9λ diameter (corresponding to 0.27 m and 100,000 unknowns) perfect electrically conducting (PEC) sphere illuminated by an 1 GHz x-polarized normally incident field. The results are compared to Mie scattering. It can be observed in Fig. 3 that the GPU results and the analytical solutions show a very good agreement.

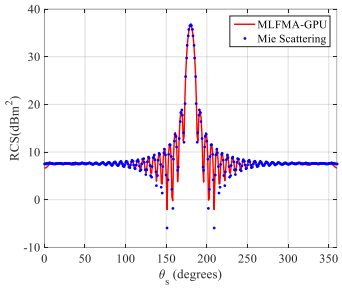


Fig. 3. RCS of a 9λ diameter PEC sphere.

B. Implementation performance on GPU cluster

In the first experiment, our GPU implementation is evaluated using the fixed-workload model (Amdahl's Law). A 22.4λ diameter PEC sphere (650K unknowns) is chosen such that it demands the use of at least 7 nodes to satisfy the required memory. Two metrics are used for the performance evaluation: speed up and scalability. The speed up is defined as the ratio of time required by multi-node GPU implementation with respect to the 7-node CPU implementation. Scalability is the normalized speedup of multiple nodes in reference to the speedup of 7 nodes. As shown in Fig. 4, the speedup factor increases from 23.7 for 7 nodes to 37 for 12 nodes. Since each node processes less workload, the GPU execution time decreases as the number of nodes increases. The inter-node communication overhead results in the difference between the speedup of total execution time and computation time. For 7 computing nodes, the speed-up for the near-field system matrix is over 86 (CPU computation time: 848s, GPU computation time: 9.5s), while the speed-up of the BICGstab iterative solution is over 22 times for 100 iterations, which is restricted by the overhead communication between computing nodes (CPU computation time: 9100s, GPU

computation time: 415.1s).

In order to investigate the scalability of this implementation, we compare how the speedup improves with increasing number of computing nodes as we keep the problem size constant, as observed in Fig. 5. The computation speedup scales similar to the theoretical linear behavior, demonstrating our efficient hardware implementation. The total speedup scales closely to the theoretical expectation demonstrating our efficiency in reducing the inter-node communication overhead.

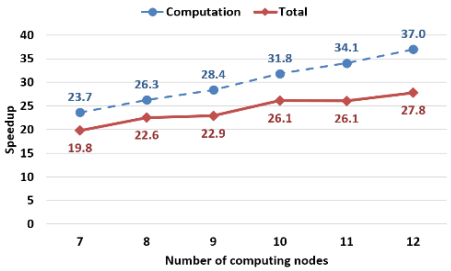


Fig. 4. Speedup analysis for the fixed-workload model (vs. 7 nodes CPU implementation, 100 iterations). Computational CPU exec time = 5573 sec, total CPU exec time = 5627 sec.

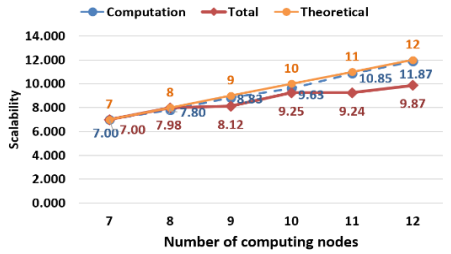
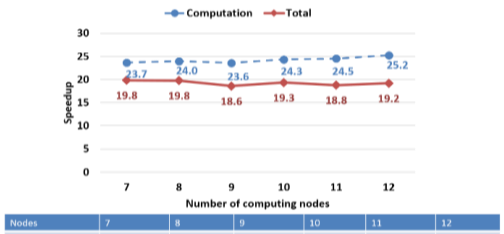


Fig. 5. Scalability analysis for the fixed-workload model.

In the second experiment, we investigate the largest problem size our GPU implementation can handle. As the number of nodes increases, the problem size is also increased so that the GPU memory in each node in fully utilized. As shown in Fig. 6, the GPU implementation can process a maximum problem size of 1.1 M unknowns with a speed up factor of 25.2.



Nodes	7	8	9	10	11	12
Size (Radius/Unknowns	11.2λ/0.65M	12.1λ/0.7M	12.5λ/0.81M	13λ/0.88M	13.4λ/0.93M	13.9λ/1.1M

Fig. 6. Speedup analysis when the number of nodes increases along with problem size increases (vs. multi-node CPU, 100 iterations).

VI. CONCLUSION

In this paper, the GPU implementation of MLFMA for electromagnetic scattering problems up to 1.1 million unknowns using our 12-node GPU cluster is demonstrated. The maximum problem size is determined by the available on-board GPU memory. For the same degree of accuracy, the GPU implementation outperforms the CPU implementation. Moreover, the GPU implementation has a good scalability as the number of computing nodes increases.

REFERENCES

[1] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Science*, vol. 31, no. 5, pp. 1225-1251, 1996.

[2] F. X. Canning, "The impedance matrix localization (IML) method for moment-method calculations," *IEEE Ant. Prop. Mag.*, vol. 32, no. 5, pp. 18-30, 1990.

[3] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Propagat. Mag.*, vol. 35, no. 3, pp. 7-12, June 1993.

[4] J. M. Song and W. C. Chew, "Multilevel fast multipole algorithm for solving combined field integral equations of electromagnetic scattering," *Microw. Opt. Tech. Lett.*, vol. 10, pp. 14-19, Sep. 1995.

[5] O. Ergul and L. Gurel, "Efficient parallelization of the multilevel fast multipole algorithm for the solution of large-scale scattering problems," *IEEE Trans. Antennas Propag.*, vol. 56, no. 8, pp. 2335-2345, Aug. 2008.

[6] X.-M. Pan, W.-C. Pi, M.-L. Yang, Z. Peng, and X.-Q. Sheng, "Solving problems with over one billion unknowns by the MLFMA," *Antennas and Propaga. IEEE Trans. on*, vol. 60, no. 5, pp. 2571-2574, 2012.

[7] J. Guan, S. Yan, and J.-M. Jin, "An OpenMP-CUDA implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-GPU computing systems," *Antennas and Propaga., IEEE Trans. on*, vol. 61, no. 7, pp. 3607-3616, 2013.

[8] S. M. Rao, D. R. Wilton, and A. W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. Antennas Propag.*, vol. AP-30, no. 3, pp. 409-418, May 1982.

[9] Q. M. Nguyen, V. Dang, O. Kilic, and E. El-Araby, "Parallelizing fast multipole method for large-scale electromagnetic problems using GPU clusters," *Antennas and Wireless Propagation Letters, IEEE*, vol. 12, pp. 868-871, 2013.

[10] V. Dang, Q. Nguyen, and O. Kilic, "Fast multipole method for large-scale electromagnetic scattering problems on GPU cluster and FPGA-accelerated platforms," *Applied Computational Electromagnetics Society Journal*, vol. 28, no. 12, 2013.

APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY EXPRESS JOURNAL
http://aces-society.org

INFORMATION FOR AUTHORS

PUBLICATION CRITERIA

Each paper is required to manifest some relation to applied computational electromagnetics. **Papers may address general issues in applied computational electromagnetics, or they may focus on specific applications, techniques, codes, or computational issues.** While the following list is not exhaustive, each paper will generally relate to at least one of these areas:

1. **Code validation.** This is done using internal checks or experimental, analytical or other computational data. Measured data of potential utility to code validation efforts will also be considered for publication.

2. **Code performance analysis.** This usually involves identification of numerical accuracy or other limitations, solution convergence, numerical and physical modeling error, and parameter tradeoffs. However, it is also permissible to address issues such as ease-of-use, set-up time, run time, special outputs, or other special features.

3. **Computational studies of basic physics.** This involves using a code, algorithm, or computational technique to simulate reality in such a way that better, or new physical insight or understanding, is achieved.

4. **New computational techniques** or new applications for existing computational techniques or codes.

5. **"Tricks of the trade"** in selecting and applying codes and techniques.

6. **New codes, algorithms, code enhancement, and code fixes.** This category is self-explanatory, but includes significant changes to existing codes, such as applicability extensions, algorithm optimization, problem correction, limitation removal, or other performance improvement. **Note: Code (or algorithm) capability descriptions are not acceptable, unless they contain sufficient technical material to justify consideration.**

7. **Code input/output issues.** This normally involves innovations in input (such as input geometry standardization, automatic mesh generation, or computer-aided design) or in output (whether it be tabular, graphical, statistical, Fourier-transformed, or otherwise signal-processed). Material dealing with input/output database management, output interpretation, or other input/output issues will also be considered for publication.

8. **Computer hardware issues.** This is the category for analysis of hardware capabilities and limitations of various types of electromagnetics computational requirements. Vector and parallel computational techniques and implementation are of particular interest.

Applications of interest include, but are not limited to, antennas (and their electromagnetic environments), networks, static fields, radar cross section, inverse scattering, shielding, radiation hazards, biological effects, biomedical applications, electromagnetic pulse (EMP), electromagnetic interference (EMI), electromagnetic compatibility (EMC), power transmission, charge transport, dielectric, magnetic and nonlinear materials, microwave components, MEMS, RFID, and MMIC technologies, remote sensing and geometrical and physical optics, radar and communications systems, sensors, fiber optics, plasmas, particle accelerators, generators and motors, electromagnetic wave propagation, non-destructive evaluation, eddy currents, and inverse scattering.

Techniques of interest include but not limited to frequency-domain and time-domain techniques, integral equation and differential equation techniques, diffraction theories, physical and geometrical optics, method of moments, finite differences and finite element techniques, transmission line method, modal expansions, perturbation methods, and hybrid methods.

Where possible and appropriate, authors are required to provide statements of quantitative accuracy for measured and/or computed data. This issue is discussed in "Accuracy & Publication: Requiring, quantitative accuracy statements to accompany data," by E. K. Miller, ACES Newsletter, Vol. 9, No. 3, pp. 23-29, 1994, ISBN 1056-9170.

SUBMITTAL PROCEDURE

All submissions should be uploaded to ACES server through ACES web site (http://aces-society.org) by using the upload button, Express Journal section. Only pdf files are accepted for submission. The file size should not be larger than 6MB, otherwise permission from the Editor-in-Chief should be obtained first. Automated acknowledgment of the electronic submission, after the upload process is successfully completed, will be sent to the corresponding author only. It is the responsibility of the corresponding author to keep the remaining authors, if applicable, informed. Email submission is not accepted and will not be processed.

EDITORIAL REVIEW

In order to ensure an appropriate level of quality control, papers are peer reviewed. They are reviewed both for technical correctness and for adherence to the listed guidelines regarding information content and format.

PAPER FORMAT

Only camera-ready electronic files are accepted for publication. The term **"camera-ready" means that the material is neat, legible, reproducible, and in accordance with the final version format listed below.**

The following requirements are in effect for the final version of an ACES Express Journal paper:

1. The paper title should not be placed on a separate page. The title, author(s), abstract, and (space permitting) beginning of the paper itself should all be on the first page. The title, author(s), and author affiliations should be centered (center-justified) on the first page. The title should be of font size 14 and bolded, the author names should be of font size 12 and bolded, and the author affiliation should be of font size 10 (regular font, neither italic nor bolded).

2. An abstract is required. The abstract should be a brief summary of the work described in the paper. It should state the computer codes, computational techniques, and applications discussed in the paper (as applicable) and should otherwise be usable by technical abstracting and indexing services. The word "Abstract" has to be placed at the left margin of the paper, and should be bolded and italic. It also should be followed by a hyphen (–) with the main text of the abstract starting on the same line.

3. All section titles have to be centered and all the title letters should be written in caps. The section titles need to be numbered using roman numbering (I. II. ….)

4. Either British English or American English spellings may be used, provided that each word is spelled consistently throughout the paper.

5. Internal consistency of references format should be maintained. As a guideline for authors, we recommend that references be given using numerical numbering in the body of the paper (with numerical listing of all references at the end of the paper). The first letter of the authors' first name should be listed followed by a period, which in turn, followed by the authors' complete last name. Use a comma (,) to separate between the authors' names. Titles of papers or articles should be in quotation marks (" "), followed by the title of the journal, which should be in italic font. The journal volume (vol.), issue number (no.), page numbering (pp.), month and year of publication should come after the journal title in the sequence listed here.

6. Internal consistency shall also be maintained for other elements of style, such as equation numbering. Equation numbers should be placed in parentheses at the right column margin. All symbols in any equation have to be defined before the equation appears or right immediately following the equation.

7. The use of SI units is strongly encouraged. English units may be used as secondary units (in parentheses).

8. Figures and tables should be formatted appropriately (centered within the column, side-by-side, etc.) on the page such that the presented data appears close to and after it is being referenced in the text. When including figures and tables, all care should be taken so that they will appear appropriately when printed in black and white. For better visibility of paper on computer screen, it is good to make color figures with different line styles for figures with multiple curves. Color should also be tested to insure their ability to be distinguished after black and white printing. Avoid the use of large symbols with curves in a figure. It is always better to use different line styles such as solid, dotted, dashed, etc.

9. A figure caption should be located directly beneath the corresponding figure, and should be fully justified.

10. The intent and meaning of all text should be clear. For authors who are not masters of the English language, the ACES Editorial Staff will provide assistance with grammar (subject to clarity of intent and meaning). However, this may delay the scheduled publication date.

11. Unused space should be minimized. Sections and subsections should not normally begin on a new page.

ACES reserves the right to edit any uploaded material, however, this is not generally done. It is the author(s) responsibility to provide acceptable camera-ready files in pdf and MSWord formats. Incompatible or incomplete files will not be processed for publication, and authors will be requested to re-upload a revised acceptable version.

COPYRIGHTS AND RELEASES

Each primary author must execute the online copyright form and obtain a release from his/her organization vesting the copyright with ACES. Both the author(s) and affiliated organization(s) are allowed to use the copyrighted material freely for their own private purposes.

Permission is granted to quote short passages and reproduce figures and tables from an ACES Express Journal issue provided the source is cited. Copies of ACES Express Journal articles may be made in accordance with usage permitted by Sections 107 or 108 of the U.S. Copyright Law. The consent does not extend to other kinds of copying, such as for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. The reproduction of multiple copies and the use of articles or extracts for commercial purposes require the consent of the author and specific permission from ACES. Institutional members are allowed to copy any ACES Express Journal issue for their internal distribution only.

PUBLICATION CHARGES

There is a $200 basic publication charge assigned to each paper for ACES members, and $300 charge for non-ACES members. Corresponding authors should be active members of the society at the time of submission and the time of publication in order to receive the reduced charge.

ACES Express Journal doesn't allow for more than four pages. All authors must comply with the page limitations. ACES Express Journal is an online journal, and printed copies are not available.

Upon completion of its first year, ACES Express Journal will be abstracted in INSPEC, in Engineering Index, DTIC, Science Citation Index Expanded, the Research Alert, and to Current Contents/Engineering, Computing & Technology.