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.