GPU implementation of the Modified Equivalent Current Approximation (MECA) method

Luis E. Tirado[1], José Á. Martínez-Lorenzo[1], Borja González-Valdés[1], Carey Rappaport[1], Oscar Rubiños-López[2], Hipólito Gómez-Sousa[2]

[1] Department of Electrical Engineering
Northeastern University, Boston, MA 02115, USA
{ltirado, jmartine, bgonzale, rappapor}@ece.neu.edu

[2] Department of Signal Theory and Communications
University of Vigo, ETSI de Telecomunicación, Campus Universitario, E-36310 Vigo, Spain
{oscar, hgomez}@com.uvigo.es

Abstract ─ This paper investigates two different methods of implementing the Modified Equivalent Current Approximation (MECA) method using CUDA parallel programing and computing platform [1]. The MECA method allows the analysis of dielectric and lossy geometries and reduces to the well-studied Physical Optics (PO) formulation in case of PEC caterers [2]. We discuss the implementation details and performance of using both an add-on toolbox for MATLAB™ to offload computations to the GPU, as well as porting MECA code to CUDA directly. We show through simulations that both methods are effective at significantly reducing the MECA algorithm computation time.

Index Terms ─ CUDA, GPGPU, MECA, parallel programming, Physical Optics.

I. INTRODUCTION

There are various methods to compute the effects of a wave scattered from arbitrary objects. Full-wave methods, like the Method of Moments (MoM) are very precise, but computationally intensive. Physical Optics methods, which approximate currents by a tangent plane method, are less accurate but faster. MECA is a good compromise, calculating equivalent currents based on oblique incidence of a plane wave on the interface together with a field decomposition into transverse electric (TE) and transverse magnetic (TM) components [3].

The current implementation of MECA using CPUs carries a heavy computational load when evaluated at multiple frequencies and observation points in different aspect angles and incident directions. However, with the recent rise in availability of Graphics Processing Unit (GPU) computing, the most processing intensive parts of the MECA code can be evaluated exactly without any approximations or interpolations at higher speeds. Parallelization using GPUs is desired for the MECA algorithm given that the algorithm is being implemented as a forward model in a personnel screening portal-based real-time whole body imaging system [4]. The Department of Homeland Security (DHS), through its mission of preventing terrorism and enhancing security, requires a high throughput, accurate, and quick detection of person-borne threats in highly secure areas. DHS calls for a security checkpoint throughput of 200-250 persons/hour. For this reason, it is essential to be able to model the scattered electric and magnetic fields from the person under test as fast as possible.

In order to reduce the total runtime of MECA, two distinct approaches have been developed. Our first approach consists of using AccelerEyes Jacket [5] GPU engine for MATLAB™ to create a vectorized version of the existing MECA code. Jacket automatically wraps MATLAB™ code into a GPU compatible form, allowing a programmer to extend existing code to parallel processing with minimal effort. The second approach is to port

Submitted On: Jan. 19, 2012
Accepted On: July 22, 2012

existing MECA code in C directly to CUDA. This paper will detail the implementation details and performance of both methods.

The paper is divided as follows. Section II gives an overview of the MECA method and briefly describes the platform used for computation. In Section III, the GPU architecture and two approaches to speed up the MECA code are discussed, while Section IV details the performance results obtained.

II. MECA OVERVIEW

The MECA method described in [2] and [3] calculates the currents from scattering objects that need not be perfect electric conductors (PEC). The objects may be dielectrics or even lossy, and MECA provides comparable results to full wave methods such as Method of Moments (MoM), except at grazing angles, where diffraction effects become more pronounced. The part of most concern in this paper is the calculation of the electric and magnetic fields given the inputs of magnetic and electric currents, observation directions, and the faceted object geometry representation, as this is the most time consuming part of the algorithm.

The MECA algorithm calculates the scattered electric field $\boldsymbol{E_k^s}$ at the observation point P_{obs} as the sum of the contributions of all the facets *i* of a given mesh geometry as [8]:

$$\boldsymbol{E_k^s} = \frac{j}{2\lambda}\sum_i \frac{e^{-jk_1 r_{ik}}}{r_{ik}}\left[\boldsymbol{E_{ik}^a} - \boldsymbol{\eta_1 H_{ik}^a} \times \hat{\boldsymbol{r}}_{\boldsymbol{ik}}\right], \quad (1)$$

where λ is the wavelength used, j is the imaginary unit, k_1 is the wavenumber of the first medium, $\boldsymbol{E_{ik}^a}$ and $\boldsymbol{H_{ik}^a}$ are the electric and magnetic fields at the observation vector $\boldsymbol{r_k}$ as defined in [3], $\boldsymbol{\eta_1}$ is the intrinsic impedance of the first medium, and $\boldsymbol{r_{ik}} = r_{ik}\hat{\boldsymbol{r}}_{\boldsymbol{ik}}$ is the position vector from the *i*-th facet centroid $\boldsymbol{r_i}$ to the observation vector $\boldsymbol{r_k}$. The magnetic field is calculated in a similar manner. Figure 1 denotes the notation of the position vectors used for an oblique wave incidence in a faceted geometry.

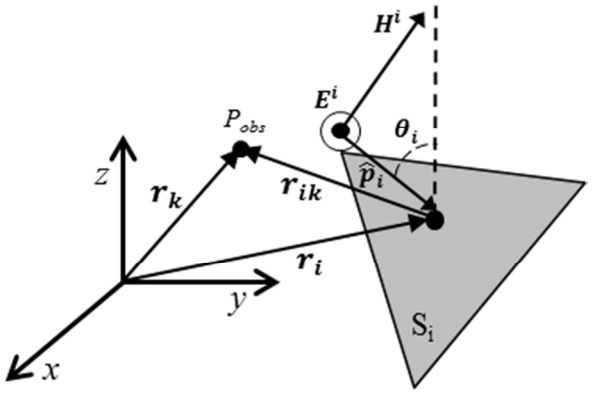


Fig. 1. Position vectors, observation point P_{obs}, propagation vector $\hat{\boldsymbol{p}}_{\boldsymbol{i}}$, angle of incidence θ_i, incident electric and magnetic fields, and facet S_i.

III. GPU IMPLEMENTATION DETAILS

A. Fermi CUDA architecture

The Fermi parallel architecture in the Tesla C2070 GPU consists of a Single Instruction Multiple Data (SIMD) processor with 14 Streaming Multiprocessors (SM). Each SM design contains 32 Streaming Processors (SPs), also called CUDA cores, 32,768 registers and 64 KB of RAM with a configurable partitioning of shared memory and L1 cache [7]. Each SM can run a variable number of threads, and the local resources are divided among them [1]. A thread on the GPU is a basic element of the data to be processed. Threads are grouped into blocks which can contain 64 to 1024 threads. Blocks are grouped together into a grid. A kernel is a code function that is executed by the CUDA device using the number of specified blocks and threads (see Fig. 2).

The CPU and GPU maintain their own DRAM and address spaces, respectively called host and device memory. Device memory can be of different types: global, shared, and constant. Table 1 lists the differences between these memory types.

Table 1: CUDA memory types

Memory	Scope	Lifetime	Access
global	grid	application	slow read/write
shared	block	kernel	fast read/write
constant	grid	application	cached read only

A challenge in developing MECA on NVIDIA CUDA enabled GPUs is making the most effective use of the platform's memory system and

resources. In addition, the productivity of GPUs under different programming paradigms can be significant depending on the application [8], which brings forth the two subsequent approaches.

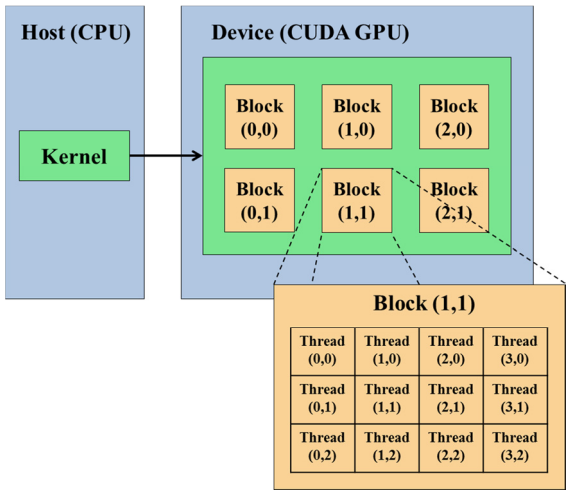


Fig. 2. CUDA Threading model.

B. JACKET GPU engine implementation

The first approach to parallelizing the existing MATLAB code is to vectorize the arithmetic operations. MATLAB and GPU computing both tend to perform best on vectorized code. The same is also true of Jacket, especially for element-wise operations which are performed just-in-time, *i.e.* they are batched together and performed in a single kernel.

Once the code is vectorized, input data is cast to Jacket's GPU data structure, allowing real-time compile-on-the-fly calculations and memory management on the GPU behind-the-scenes. Functions called on GPU data execute on the GPU automatically without any extra programming.

As an example, assume that there are 80,000 facets (`nT`), and 181 observation points (`nr`) in the polar angle $\boldsymbol{\theta}$ sweep for a single circumferential angle $\boldsymbol{\phi}$ in our simulation. The single-threaded MATLAB implementation relies on a loop that iterates over the observation points, calculating the electric and magnetic field x, y, and z components for each point. The Jacket GPU implementation, however, reshapes the intermediate data quantities into single matrices of size (`3, nT*nr`), which eliminates looping, and many GPU threads work at the same time to compute the output fields in steps. The end result is that the GPU multiprocessor occupancy is increased up to 72% for the example case, and thus, the vectorized code runtime is reduced as compared to the single-threaded for-loop version of the code.

One disadvantage of this type of vectorization is that, for the example case, up to 2.5 GB of GPU memory is allocated to perform the calculations. In order to be able to process a larger data set, the algorithm needs to be broken down into chunks, and many intermediate calculations are repeated. The data chunks then have to be arranged to match the original output format. Thus, for each additional $\boldsymbol{\phi}$ cut in our example, the computational time is doubled due to memory constraints. Even though the MECA runtime is reduced with the Jacket code, there is still room for improvement in reducing memory usage and maximizing GPU resource occupancy.

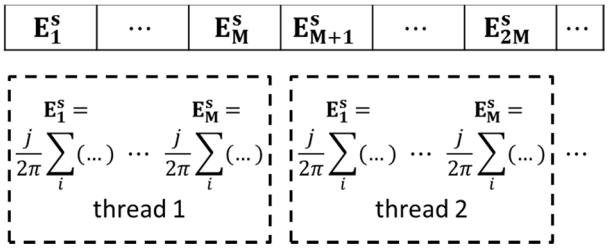
C. CUDA C code implementation

It was advantageous to reuse the existing structure of an already existing and validated version of the MECA code ported to C [2]. In the OpenMP C version of MECA, the sum in Eq. (1) is calculated in series by use of an accumulating variable. Figure 3(a) shows the implementation of the scattered field computations for the OpenMP C version of the code.

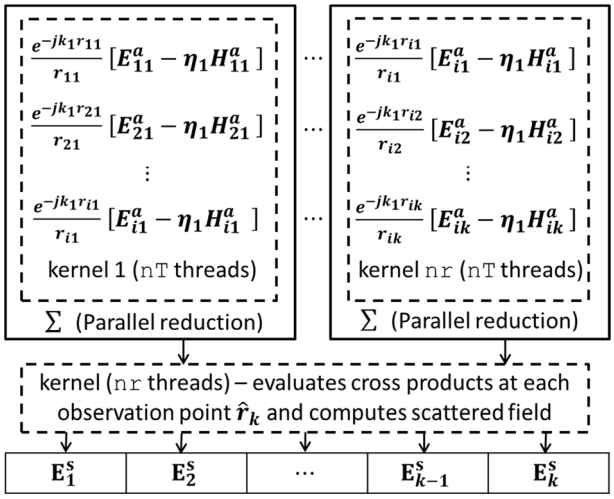
In the CUDA version of MECA, changes must be made to account for the GPU architectural differences to avoid having many threads writing to the same memory address, resulting in a race condition. Without modifying the C code, atomic operations would be necessary to compute the sum in Eq. (1). Atomic operations read, modify, and write back a value to memory without interfering with other threads. However, the GPU cannot perform many atomic operations without considerable delays. In our example with 80,000 facets, the computations using atomic operations are one order of magnitude slower than the single threaded MATLAB implementation of MECA.

The CUDA implementation strategy is shown in Fig. 3(b). In order to avoid atomic operations, we structure the code to use one GPU thread to do intermediate calculations for each facet in the input geometry. For our example from the previous section, 256 threads per block are instantiated, so we end up using `ceil(nT/256) = 313` blocks. Given that the current NVIDIA Fermi architecture allows for up to 65,536 blocks of up to 1024 threads each, it is clear that a large number of facets can be

evaluated in parallel limited only by GPU memory constraints.



a) Each thread computes for a set of observation points (OpenMP C approach)



b) nT threads compute part of Eq. (1) in nr kernels, parallel reduction, last kernel (nr threads) computes field (CUDA C approach)

Fig. 3. OpenMP & CUDA MECA implementations

More explicitly, the CUDA code begins by copying the variables used by the algorithm into the GPU's global memory. Next, nr kernels are launched, in which each of the threads calculates part of the *i*-th term inside the sum in Eq. (1) independently and writes each result to a different index of an array initialized in GPU memory. Subsequently, a parallel reduction algorithm [9] is used to sum all the facet contributions for the *k*-th observation point. This algorithm takes advantage of the most efficient implementation of a parallel sum reduction automatically based on data input size, relying on use of fast shared memory [10]. Lastly, a kernel with nr threads loads the previously-saved reductions for each of the coordinate axes. Next, it evaluates the cross-products of Eq. (1) and the analogous magnetic field equation to compute the real and imaginary parts of the scattered electric and magnetic fields at each observation point $\boldsymbol{r}_k$.

Comparing to the Jacket version, in the example case, only 25 MB of GPU memory is required to perform the computations, and for each additional observation point, only 24 extra bytes of memory are required to be allocated on the GPU, which greatly falls below the memory requirements for the vectorized Jacket version of the code.

Based on the fact that the brute force computation of the scattered far fields is an O(nT*nr) operation, there is a massive amount of parallelism that is exploited by porting the MECA code to CUDA enabled GPUs.

IV. RESULTS

In addition to computing the total runtime results for the Jacket and CUDA implementations, we also compare these to MATLAB code (single threaded) [3] and OpenMP multi-threaded C code developed in [2]. The Jacket and CUDA implementations are also validated against the existing MATLAB and C codes for numerical accuracy. The MECA code has been widely validated with other electromagnetic codes in previous works, [2,3] which compare MECA with other electromagnetic methods. The discussion on accuracy is out of the scope of this paper, which focuses on the speed improvements.

The simulations conducted in this paper are performed using a single workstation 2.8 GHz Intel® Core™ i7 930 quad-core CPU with an NVIDIA Tesla™ C2070 Computing Processor with 6 GB of GDDR5 VRAM. The Tesla™ C2070 contains 448 stream processors running at 1.15 GHz, which has a double precision floating point peak performance of 515 GFLOPs. MATLAB version 7.11.0.584 (R2010b) is used along with Jacket version 1.8.1. The compiled CUDA code and Jacket 1.8.1 both are based on NVIDIA's CUDA version 4.0.

A square plate geometry is used for the performance tests, by varying the number of facets and keeping the number of observations fixed to 722 to compare the performance to the results presented in [2]. The number of facets is kept fixed at 80,000 while the number of observations is varied. The maximum number of facets used in the tests is 8×10^6, and the maximum number of

observations points is 360 θ cuts × 360 ϕ cuts. The plane incident wave frequency is 60 GHz.

Figures 4 and 5 show the runtime for computing Eq. (3) as a function of the number of facets for the CUDA version of the code versus the MATLAB and OpenMP C versions for near and far-fields, respectively. Jacket M code timing results are not included in these figures due to a product limitation in the implementation of the Kronecker tensor product in the Jacket version used to construct vectorized data matrices with more than 1.67×10^7 elements in one dimension.

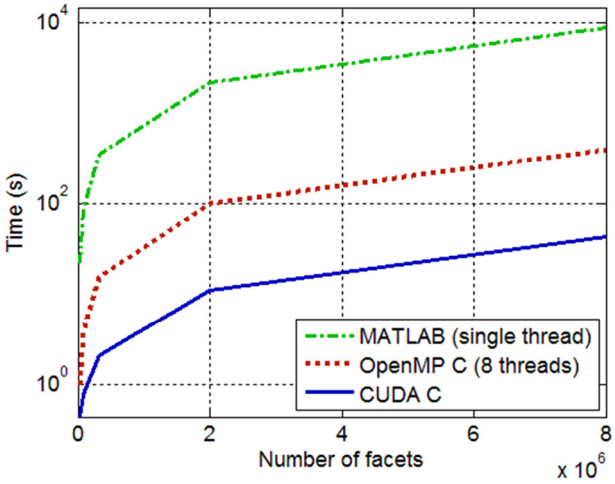


Fig. 4. Runtime vs. number of facets for near-field calculations (722 observations).

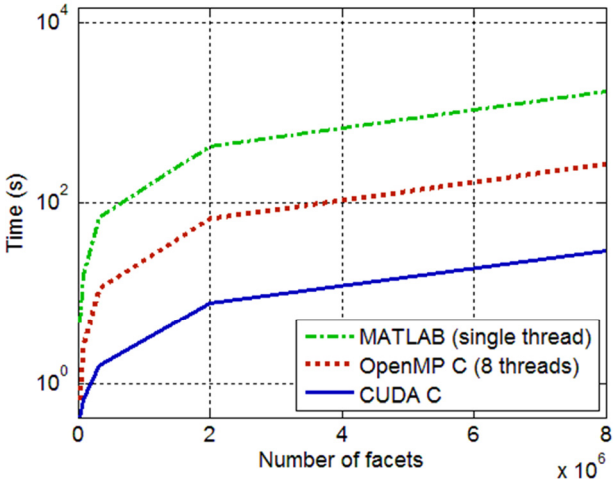


Fig. 5. Runtime vs. number of facets for far-field calculations (722 observations).

The CUDA C implementation is 1.5 to 9.3 times faster than the OpenMP C implementation with GPU multiprocessor occupancy varying from 70 to 100% as a function of the number of facets used in the computations. The algorithm reaches 100% GPU occupancy at 2.9×10^6 facets and 4.5×10^6 facets for the near and far-field versions, respectively.

Figures 6 and 7 show the runtime for computing Eq. (1) for 80,000 facets and a varying number of observations for the MATLAB (single-threaded), OpenMP (multi-threaded), Jacket M, and CUDA C codes for near and far-fields, respectively. From the timing results obtained, we can ascertain that CUDA C code scales linearly with the number of observations in the same way as the MATLAB and OpenMP C versions do.

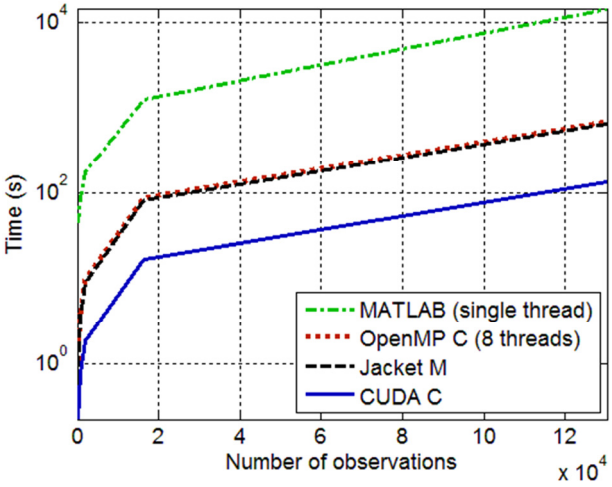


Fig. 6. Runtime vs. number of observations for near-field calculations (80,000 facets).

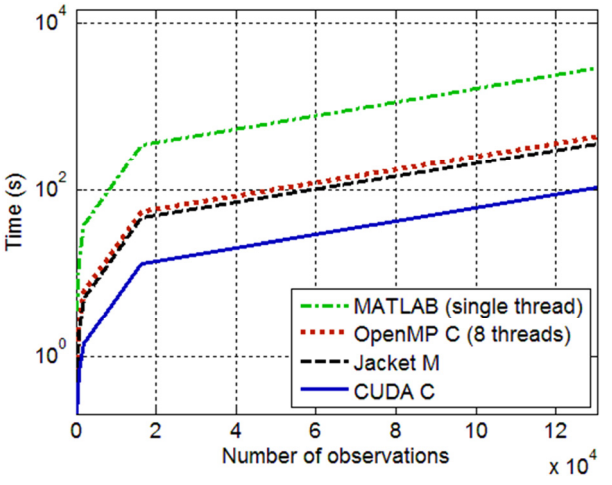


Fig. 7. Runtime vs. number of observations for far-field calculations (80,000 facets).

The Jacket M implementation for 80,000 facets is only 7% to 18% faster than the OpenMP C implementation. This is because the input data needs to be processed in chunks due to the high memory usage required by the vectorization. In addition, the Jacket code GPU multiprocessor occupancy is 72% or 68% for the near and far-field implementations, respectively. The CUDA version for the same number of facets, however, is 80% or 84% efficient for the near and far-fields. It computes all of the data in a single run, which makes it 3.8 to 5.3 times faster than the OpenMP code, a marked performance improvement.

To validate the numerical accuracy of the CUDA implementation, the maximum error between the OpenMP C and CUDA C results is computed for both the near and far-field cases with varying facet numbers from the earlier simulation. The results are shown in Table 2 and Table 3 for near and far-field observations, respectively.

Table 2: CUDA and OpenMP near-field maximum error

Number of facets	Total electric field maximum error (V/m)	Total magnetic field maximum error (A/m)
20000	8.39E-14	2.35E-16
80000	1.07E-13	2.80E-16
320000	1.94E-13	4.85E-16
2000000	6.30E-12	1.68E-14
8000000	8.85E-12	2.37E-14

Table 3: CUDA and OpenMP far-field maximum error

Number of facets	Total electric field maximum error (V/m)	Total magnetic field maximum error (A/m)
20000	2.61E-11	6.92E-14
80000	6.13E-11	1.63E-13
320000	1.23E-10	3.27E-13
2000000	8.43E-10	2.24E-12
8000000	3.82E-09	1.01E-11

The variation of the maximum error difference is due to the parallel sum reduction algorithm used to sum all the facet contributions for each observation point, whereas the OpenMP code adds up the contributions serially. Given that floating point operations are non-commutative, small differences between the CPU and GPU results are expected.

V. CONCLUSION

This paper presents a CUDA version of a modified PO method known as the modified equivalent current approximation (MECA), which is valid for both PEC and dielectric objects. Our results show that the computational performance of the CUDA version is increased up to 9.3 times with respect to the OpenMP C algorithm timings. This shows promise to implement an inverse reconstruction algorithm, taking advantage of the speedup and the excellent numerical accuracy that the CUDA platform provides.

ACKNOWLEDGMENT

This material is based upon work supported by the U.S. Department of Homeland Security under Award Number 2008-ST-061-ED0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the U.S. Department of Homeland Security.

REFERENCES

[1] M. Ujaldon, "Using GPUs for Accelerating Electromagnetic Simulations," ACES Journal, vol. 25, no. 4, pp. 294-302, 2010.

[2] H. Gómez-Sousa, J. A. Martínez-Lorenzo, O. Rubiños-López, J. G. Meana, M. Graña-Varela, N. Gonzalez-Valdes, M. Arias-Acuña, "Strategies for Improving the Use of the Memory Hierarchy in an Implementation of the Modified Equivalent Current Approximation (MECA) Method," ACES Journal, vol. 25, no. 10, pp. 841-852, 2010.

[3] J. G. Meana, J. A. Martinez-Lorenzo, F. Las-Heras, and C. Rappaport, "Wave Scattering by Dielectric and Lossy Materials using the Modified Equivalent Current Approximation," IEEE Transactions on Antennas and Propagation, vol. 58, no. 11, pp. 3757-3761, 2010.

[4] J. L. Fernandes, C. Rappaport and D. M. Sheen, "Improved Reconstruction and Sensing Techniques for Personnel Screening in Three-Dimensional Cylindrical Millimeter-Wave Portal Scanning," Proc. SPIE 8022, 802205, 2011.

[5] AccelerEyes, Jacket, Version 1.8.1, http://www.accelereyes.com, Sep. 2011.

[6] C. A. Balanis, *Advanced Engineering Electromagnetics,* 1st ed. New York, USA: John Wiley and Sons, 1989.

[7] NVIDIA, "NVIDIA's Next Generation CUDA™ Compute Architecture: Fermi™," Version 1.1, http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2009.
[8] M. Malik, T. Li, U. Sharif, R. Shahid, T. El-Ghazawi, G. Newby, "Productivity of GPUs under Different Programming Paradigms," Concurrency and Computation: Practice and Experience, vol. 24, no. 2, pp. 179-191, 2012.
[9] J. Hoberock, N. Bell, "Thrust: A Parallel Template Library," V1.3.0, http://www.meganewtons.com, 2010.
[10] NVIDIA, "Thrust Quick Start Guide," Version 01, http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/Thrust_Quick_Start_Guide.pdf, Jan. 2011.

Luis Eladio Tirado was born in Mayagüez, Puerto Rico in 1983. He received the B.S. and M.S. degrees in Electrical Engineering from Northeastern University in 2006 and 2008, respectively. He is currently on leave of absence from Raytheon Integrated Defense Systems (IDS) to complete the Ph.D. in Electrical Engineering at Northeastern University, Boston, MA, where he is part of the Awareness and Localization of Explosives-Related Threats (ALERT) Department. His research interests include GPU implementations of forward and inverse millimeter wave models.

José Ángel Martínez-Lorenzo (S'03–M'05) was born in Madrid, Spain, in 1979. He received the M.S. and Ph.D. degrees in telecommunications engineering from the University of Vigo, Vigo, Spain, in 2002 and 2005, respectively.

He was a Teaching and Research Assistant with the University of Vigo from 2002 to 2004. He joined the faculty at the University of Oviedo, Gijón, Spain, in 2004, where he was an Assistant Professor in the area of signal theory and communications until 2006. During spring and summer 2006, he was a Visiting Researcher with the Bernard Gordon Center for Subsurface Sensing and Imaging Systems (Gordon-CenSSIS), Northeastern University, Boston, MA. He was appointed as a Research Assistant Professor with the Department of Electrical and Computer Engineering, Northeastern University. He is currently an active member of the Awareness and Localization of Explosives-Related Threats (ALERT), a Department of Homeland Security Center of Excellence, Northeastern University. He has authored over 80 technical journal and conference papers. His research is geared toward the understanding, modeling, and quantitative prediction of complex electromagnetic problems with special application to security sensing systems, communication systems, and biomedical systems.

Borja González-Valdés was born in Gijon, Spain. He received the Electrical Engineering degree and Ph.D. from the University of Vigo, Spain, in 2006 and 2010 respectively. From 2006 to 2010, he was a research grant holder and then postdoctoral researcher with the Antenna and Optical Communications group at the University of Vigo. During 2008 and 2009, he was a visiting researcher at the Gordon CenSSIS Center, Northeastern University, Boston, USA. In 2011, he joined the ALERT Center of Excellence, Northeastern University, Boston, USA as a Postdoctoral Research Associate. His research interests include antenna design, inverse scattering, advanced imaging techniques, and THz technology.

Carey M. Rappaport (S'80–M'87–SM'96–F'06) received the B.S. degree in mathematics, the B.S., M.S., and E.E. degrees in electrical engineering in 1982, and the Ph.D. degree in electrical engineering in June 1987 from the Massachusetts Institute of Technology (MIT), Cambridge.

He was a Teaching and Research Assistant with MIT from 1981 to 1987 and also with COMSAT Labs, Clarksburg, MD, and The Aerospace Corporation, El Segundo, CA, during summers. In 1987, he joined the faculty at Northeastern University, Boston, MA, where he has been a Professor of electrical and computer engineering since July 2000. During fall 1995, he was a Visiting Professor of electrical engineering with the Electromagnetics Institute, Technical University of Denmark, Lyngby, Denmark, as part of the W. Fulbright International Scholar Program. During the second half of 2005, he was a Visiting Research Scientist with the Commonwealth Scientific Industrial and Research Organization, Marsfield Australia. He has consulted for Geo-Centers, Inc., PPG, Inc., Alion Science and Technology, Inc., and several municipalities on wave propagation and modeling and also on microwave heating and safety. He was a Principal Investigator of an ARO-sponsored Multidisciplinary University Research Initiative on Humanitarian Demining and a Co-Principal Investigator of the NSF-sponsored Bernard Gordon

Center for Subsurface Sensing and Imaging Systems (Gordon-CenSSIS), Northeastern University, Boston, MA. He has authored over 300 technical journal and conference papers in the areas of microwave antenna design, electromagnetic wave propagation and scattering computation, and bioelectromagnetics. He is the holder of two reflector antenna patents, two biomedical device patents, and three subsurface sensing device patents.

Dr. Rappaport is a member of Sigma Xi and Eta Kappa Nu professional honorary societies. He received the IEEE Antenna and Propagation Society's H. A. Wheeler Award for best applications paper, as a student, in 1986.

Oscar Rubiños-López received the M.S. and Ph.D. degrees in telecommunication engineering from the Universidad de Vigo, Vigo, Spain, in 1991 and 1997, respectively. He joined the Universidad de Vigo in 1991 and is currently an associate professor with the Dept. of Signal Theory and Communications at the Universidad de Vigo. During different periods of 2004, 2005 and 2007, he was a Visiting Researcher at Chalmers University of Technology in Goteborg (Sweden). His research interests include: the analysis and design of broadband antennas, numerical simulation of applied electromagnetic problems, terahertz technology for electromagnetic sensing applications, satellite systems and wireless communications. He has coauthored over 80 technical journal and conference papers. From 2001 to 2006, he held the position of Vice-President of University Extension (2001-2002) and for University Extension and Students at the University of Vigo.

Hipólito Gómez-Sousa received the M.S. degree in telecommunications engineering from the University of Vigo, Vigo, Spain, in 2009. Since 2009, he has been with the Department of Signal Theory and Communications, University of Vigo. His current research interests are on computational electromagnetism, THz sensing systems, and quantum cryptography.