A Stacking Scheme to Improve the Efficiency of Finite-Difference Time-Domain Solutions on Graphics Processing Units

Veysel Demir

Department of Electrical Engineering
Northern Illinois University, DeKalb, IL 60115, USA
demir@ceet.niu.edu

Abstract—Advances in computer hardware technologies accompanied by easy-to-use parallel programming software platforms have led to the wide spread use of parallel processing architectures, such as multi-core central processor units (CPUs) and graphic processing units (GPUs), in technical and scientific computing. Among electromagnetic numerical analysis methods, the finite-difference time-domain (FDTD) method is very well suited for parallel programming, and several implementations of FDTD have been developed and reported to solve electromagnetics problems orders of magnitude faster. Examination of performances of these implementations reveals that, in general, it is more efficient to solve larger FDTD domains than smaller domains. In this paper it is demonstrated that one can exploit the higher efficiency inherent to the solution of larger problem sizes to solve parameter sweep and optimization problems faster: instead of solving multiple smaller FDTD domains separately, these domains can be combined or stacked to form a larger problem and the large problem can be solved more efficiently. It has been shown that up to 40% faster solution can be achieved on GPUs with this method.

Index Terms—FDTD methods, parallel architectures, graphics processing unit (GPU) programming, Compute Unified Device Architecture (CUDA), hardware accelerated computing.

I. INTRODUCTION

The finite-difference time-domain (FDTD) method [1]-[3] has been the most popular numerical analysis technique throughout the past decades to solve a variety of electromagnetics problems. In FDTD, the problem space is composed of cells, in which electric and magnetic field components are located at discrete positions. These field components are recalculated at every time-step of a time-marching algorithm. The calculations for each cell can be performed independent from other cells at each time step; thus FDTD is very suitable for parallel programming. Until recently central processor units (CPUs) have been the main hardware architecture to perform high performance scientific and technical computing, and several implementations of FDTD have been developed for high performance CPU clusters and multi-core CPUs.

Recently, graphic processing units (GPUs), equipped with hundreds of processing cores, have evolved rapidly and outmatched CPUs in terms of computation power. Accompanied by advances in parallel programming software technologies, the advances in GPUs enabled widespread use of these devices, which had been initially designed for processing computer graphics, for general purpose computing. Initially GPUs were designed to support only single-precision floating-point arithmetic operations, which is sufficient for graphics processing. To further aid general purpose computing, latest generation GPUs support double-precision floating-point arithmetic operations as well. Thus graphics cards have evolved into computation cards.

Implementations of various numerical analysis methods have been developed on GPU platforms to solve electromagnetics problems faster. In particular, several implementations of FDTD method have been developed and reported [4]-[24]. These implementations are based on various programming platforms. For instance, [4]-[7] are

based on OpenGL; [8]-[14] are based on Brook; [15] uses High Level Shader Language (HLSL); and [16]-[20] are based on compute unified device architecture (CUDA) [25]. Independent of the different programming languages used in these contributions, it has been shown that FDTD problems can be solved orders of magnitude faster on graphics cards.

Parameter sweep and optimization are two commonly used techniques in the design of circuits. Electromagnetic calculations are computationally expensive and usually it takes a long time to run a simulation. Since, parameter sweeps and most optimization techniques need a large number of runs to achieve their target; long execution times can seriously hinder the adoption of these techniques. With the significant speed gains available with the GPU based FDTD solvers; optimization becomes a viable option in electromagnetic design [13]. The use of GPU based FDTD solvers for optimization and parameter sweep has been presented in [13]. Similarly, GPU based FDTD is used in [16] and [20] in optimization for radio coverage prediction.

One reasonable way to measure the efficiency of an FDTD implementation is to calculate the number of cells processed per second, in other terms the *throughput*, such as [26]

$$NMCPS = \frac{n_{steps} \times Nx \times Ny \times Nz}{t_s} \times 10^{-6}, \qquad (1)$$

where *NMCPS* is the number of million cells processed per second, n_{steps} is the total number of time steps the program has been run, and t_s is the total computation time in seconds. Here, Nx, Ny, and Nz are the number of cells in an FDTD problem space in *x*, *y*, and *z* directions, respectively. Such throughput data as a function of the FDTD domain size have been provided in [18], [24], and [26]. Generally the trend of throughput as a function of problem size in these data is similar to that illustrated in Fig. 1. The data shows that the computation efficiency is directly proportional to the problem size; i.e. the efficiency is higher for larger problem sizes. This trend is expectable since it is more efficient to load larger amounts of data to the GPU memory at once than loading smaller chunks at multiple times. Furthermore, the multiprocessors can more efficiently schedule the threads for larger number of threads, thus problem sizes. The higher efficiency inherent to the solution of a larger domain implies that if solutions of multiple smaller problems are required, it will be more efficient to combine their spatial domains as a single larger domain and solve the larger problem. This scenario fits to optimizations or parameter sweeps very well, since solutions of multiple similar size problems are sought in such cases: Similar size FDTD spatial domains can be stacked, where each domain is electromagnetically isolated from the others, and a large domain can be obtained. The entire combined domain can be solved in a single run. Using this method, a significantly faster solution can be achieved compared to the case where all the individual domains are solved separately.

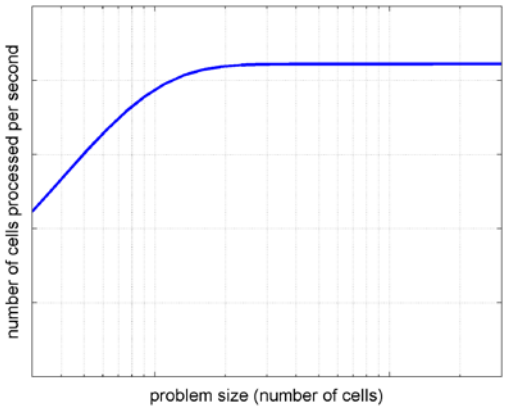

Fig. 1. Throughput versus problem size.

This paper demonstrates, via examples, that overall simulation time can be significantly reduced by a CUDA based FDTD code using the presented stacking method. The paper is organized as follows. Section II presents an FDTD implementation based on CUDA for GPU platforms. Section III discusses various schemes to stack FDTD spatial domains and shows time reductions in calculation times achieved by these stacking schemes in an example case. Section IV presents an analysis to examine the effect of orientation of the problem geometry on the efficiency of the solutions. Section V discusses some other benefits the stacking method can offer to the FDTD solutions of problems.

II. FDTD USING CUDA

New software development platforms and languages have accompanied and aided the general purpose GPU (GPGPU) computing as it has evolved and become widespread. OpenGL, Brook, and HLSL are among those languages that are commonly used, as discussed earlier, to program FDTD. However, steep learning curves of these languages prevent their widespread use.

Recently, Compute Unified Device Architecture (CUDA) is introduced by NVIDIA as a software development platform. CUDA is a general purpose parallel computing architecture. To program the CUDA architecture, developers can use C, which can then be run at great performance on a CUDA enabled processor [27]. Compared to other programming languages, some advantages and disadvantages of CUDA can be listed [28]. One of the main limitations of CUDA is that it can be used only with CUDA-enabled GPUs, which are manufactured only by NVIDIA. Some of the main advantages of CUDA are listed as availability of scattered reads from arbitrary addresses in memory, and shared memory that can be simultaneously accessed by the threads in the same thread block. Besides these advantages, two major factors led to its widespread use in many applications including FDTD: NVIDIA provides extensive support to programmers who would like to develop codes using CUDA, and programming for GPU computing is easier with CUDA.

In this current contribution a CUDA implementation of FDTD is used to prove the performance improvement achieved by the proposed stacking method. The details of this implementation are presented in [29], where the algorithm of the implementation is referred to as *xy*-mapping. In order to aid the understanding of discussions in the subsequent sections, some details of this implementation are summarized here as a reference.

In CUDA a number of threads work in parallel and form a thread block, while a number of thread blocks form a grid. In the *xy*-mapping algorithm, a grid of threads is mapped to the cells in the *xy*-plane cut of an FDTD problem space. Each thread is mapped to a cell and the field components in that cell are updated by the thread. Each thread then traverses the cells in the same column in the *z* direction and updates the field components in the same column, as illustrated in the pseudocode in Listing 1. This algorithm implies that there is anisotropy in the computations in the sense that computation in the *z* direction is treated differently.

In CUDA, it is very important to have global memory accesses coalesced to achieve faster computations. In the *xy*-maping algorithm, to make sure that the global memory accesses are coalesced, the FDTD problem space is enlarged by padding extra cells to the problem space, such that the number of cells in *x* and *y* directions are integer multiples of 16. For instance, if a problem space is composed of $Nx \times Ny \times Nz$ cells, the problem size becomes $Nxx \times Nyy \times Nz$ after the padding, where Nxx and Nyy are integer multiples of 16.

It should be noted that in the *xy*-mapping algorithm, the fields in the cells padded in the *x* direction are computed by the associated threads, while the cells padded in the *y* direction are not processed, as shown in Listing 1. This algorithm implies another anisotropy in the *x* and *y* directions throughout the computations. As a result, the FDTD algorithm in consideration is anisotropic in the sense of computations in *x*, *y*, and *z* directions. Therefore, if a non-cubic problem space will be computed, different computation performances should be expected if the problem geometry is rotated to align in different directions.

```
Function update_magnetic_fields

  Calculate thread index ti
  Calculate cell index i and j using ti

  If j < ny
    For k from 1 to nz
      Update Hx, Hy, and Hz
    End for
  End if

End function
```

Listing 1. Pseudocode of CUDA kernel to update magnetic field components based on *xy*-mapping.

III. STACKING SCHEMES

Many different scenarios can be envisioned to stack smaller FDTD spatial domains to obtain a larger domain. For instance, domains can be stacked in a linear sequence in one-dimension, a planar sequence in two-dimensions, or a cuboidal sequence in three-dimensions. In this contribution, linear stacking is considered for the analyses. The linear stacking, as well, can be achieved in three-different scenarios: stacking in the *x* direction, stacking in the *y* direction, or stacking in the *z* direction, as illustrated in Fig. 2 for three domains. These three schemes will be referred to as *x*-stacking, *y*-stacking, and *z*-stacking, respectively, in the following discussions.

As discussed in the previous section, the algorithm acts differently in different directions. This algorithmic anisotropy would cause a different calculation time every time the problem spaces are stacked in different directions.

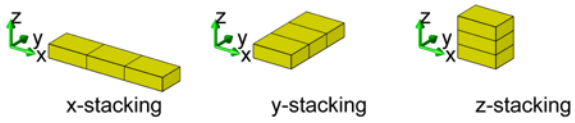

Fig. 2. FDTD problem spaces stacking schemes.

A performance analysis test is performed to find which direction is the best for stacking. An NVIDIA® Tesla™ C1060 Computing Processor running at 1.3 GHz is used for the tests presented in this paper. A problem space composed of $100\times100\times25$ cells is used as the base FDTD spatial domain, where a smaller number of cells is used in the *z* direction to model a microstrip type structure. Then this domain is stacked in *x*, *y*, and *z* directions. Each time the number of stacked domains is increased, simulation is performed, and throughput is calculated. The results of this test are plotted in Fig. 3. It is found that as the problem size increases the efficiency increases, as expected. Furthermore, the *x*-stacking is found to be the best performing scheme, while the *z*-stacking is the worst. One reason for why *x*-stacking performs better is that as the problem domain is enlarged in the *x* direction, the number of unnecessarily processed padding cells becomes negligible compared to the number of cells in the main domain. The reason for why the *y*-padding performs better than the *z*-padding is that as the domain size increases, the number of thread blocks also increases with the *y*-padding, and the thread blocks are scheduled more efficiently by the GPU multiprocessor.

In this given example, the base problem domain size is 250,000 cells, and this number of cells is processed with a throughput of 340 million cells per second. When 128 of this domain are stacked in the *x* direction, the problem size becomes 32 million cells, while the throughput becomes 497 million cells per second. These numbers show that, for instance, if 128 runs of the base domain are required for an optimization problem, it will be more than 40% faster to complete the optimization using the proposed stacking method compared to the case where all the base domains are solved separately.

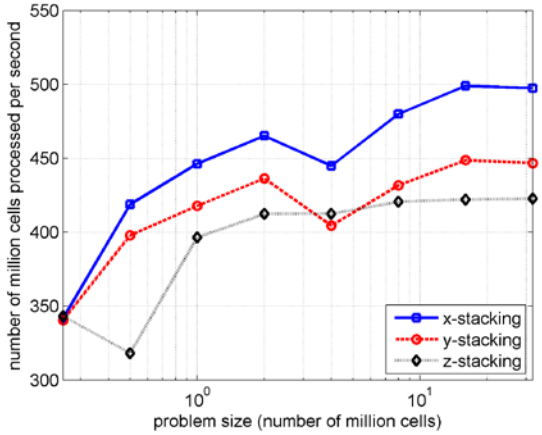

Fig. 3. Throughput of different stacking schemes.

It should be noted that these findings are valid only for the code of the presented *xy*-mapping algorithm and for different codes based on different algorithms the efficiencies due to stacking directions may be different. In any case, an increase in efficiency should be expected if the problem size is increased by stacking.

IV. ALIGNMENT OF GEOMETRIES

The analysis presented in the previous section revealed that it is better to stack the FDTD spatial domains in the *x* direction to achieve the best performance out the presented algorithm. In general a problem space can be in arbitrary size in

different directions; i.e. number of cells in a direction may be different from the number of cells in other directions. It is easy to rotate the geometry such that a side is aligned in any desired direction. For instance, one can align the longest side of a domain in *x*, *y*, or *z* direction by simple transformations.

Since there is a flexibility to align sides in desired directions, one can expect different performances from stacking for different alignments. In order to find which alignment is the best, an alignment test is performed as described below. A base FDTD domain with size of 64 cells on the short side, 128 cells on the medium side, and 192 cells on the long side, as shown in Fig. 4, is prepared. Then this domain is rotated and stacked in the *x* direction for six different alignment scenarios. For instance, Fig. 5 illustrates one of these cases, in which the three copies of the base domain are stacked in the *x* direction such that the short side is aligned in the *x* direction, and the long side is aligned in the *z* direction.

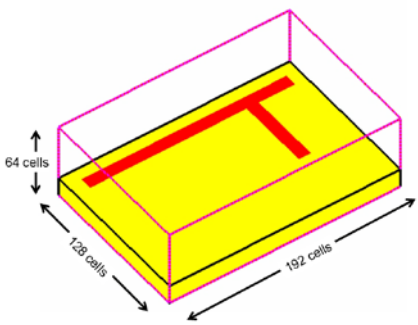

Fig. 4. A problem space with different sizes in different directions.

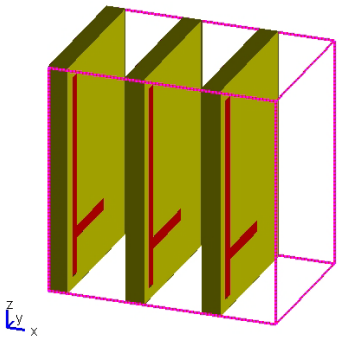

Fig. 5. Base FDTD domain stacked in the x direction such that short side is aligned in the x direction and the long side is aligned in the z direction.

For each of these six cases, first, the base domain is simulated alone and then 20 copies of the base domain are stacked and simulations are repeated. The throughput is calculated for each simulation, and results are tabulated as shown in Table 1. The results reveal that for all of these six cases the efficiencies of the stacked domains simulations are better than that of individual domains simulation. When the efficiencies of the stacked domains simulations are compared, no significant difference has been observed between the six alignments. However, the case in which the shortest side is aligned in the *z* direction, the last row in Table 1, has slightly higher throughput, thus efficiency. It should be reminded that this alignment efficiency analysis is valid for the code of the *xy*-mapping in consideration, and for a different code the results might be different. Nonetheless, alignment directions of geometries shall be taken into consideration to reach the best performance out of the proposed stacking algorithm.

V. OTHER ADVANTAGES OF STACKING

The tests presented in the previous section are performed using a simple FDTD domain with PEC boundaries and a microstrip structure excited by a single voltage source. For some other classes of problems, stacking can provide some other means to achieve better performance in speed as well as memory usage.

One class of problems that can benefit from stacking is scattering due to an incident field. For scattering calculations, the problem space is illuminated by an incident field that has to be recalculated at every time step of time-marching. In an optimization problem, all problem spaces will be excited with the same incident field. Therefore, if incident field is calculated and stored for the base domain, it can be used to excite the other domains in the stack as well. This way, recalculation and storage of fields for separate domains can be avoided and efficiency can be significantly improved both in terms of simulation time and memory.

Another class of problems is the calculation of scattering parameters in a multi-port circuit. For the solution of such problems, in each simulation,

one port is active as a source and scattering parameters are calculated with respect to the active port. Thus for an *N*-port problem, the calculation shall be repeated *N* times. These *N* problem spaces can be stacked and simulated at one run; thus a faster solution can be achieved. Another advantage is that, all FDTD updating coefficients are essentially the same in all of these individual problems. Therefore, it is sufficient to calculate and store the updating coefficients only for a base domain and reuse these coefficients in other domains. Thus efficiency can be achieved in terms of memory use as well.

Table 1. Efficiency of stacking with respect to alignment.

direction of alignment (x, y or z)	short side	medium side	long side	number of stacked domains	stacked domain size (million cells)	number of million cells processed per second
	x	y	z	1	1.6	324
	x	y	z	20	31.5	499
	x	z	y	1	1.6	375
	x	z	y	20	31.5	496
	y	x	z	1	1.6	344
	y	x	z	20	31.5	480
	y	z	x	1	1.6	414
	y	z	x	20	31.5	495
	z	x	y	1	1.6	436
	z	x	y	20	31.5	499
	z	y	x	1	1.6	448
	z	y	x	20	31.5	503

VI. CONCLUSION

The concept of stacking FDTD problem spaces to achieve computation efficiency in terms of solution speed is introduced for optimization and parameter sweep problems on graphics processing platforms. In particular, an FDTD implementation based on CUDA is discussed for GPU platforms and it has been shown that significantly shorter solution times can be achieved if problem spaces are stacked and solved at one run compared to the case where all these problems are solved separately. It has also been shown that, for some classes of problems, stacking can achieve memory efficiency as well.

REFERENCES

[1] K. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," *IEEE Transactions on Antennas and Propagation*, vol. 14, pp. 302–307, May 1966.

[2] A. Taflove and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 3rd edition, Artech House, 2005.

[3] A. Elsherbeni and V. Demir, *The Finite Difference Time Domain Method for Electromagnetics: With MATLAB Simulation*s, SciTech Publishing, 2009.

[4] S. E. Krakiwsky, L. E. Turner, and M. M. Okoniewski, "Graphics Processor Unit (GPU) Acceleration of Finite-Difference Time-Domain (FDTD) Algorithm," *Proc. 2004 International Symposium on Circuits and Systems*, vol. 5, pp. V-265–V-268, May 2004.

[5] S. E. Krakiwsky, L. E. Turner, and M. M. Okoniewski, "Acceleration of Finite-Difference Time-Domain (FDTD) Using Graphics Processor Units (GPU)," *2004 IEEE MTT-S International Microwave Symposium Digest*, vol. 2, pp. 1033–1036, Jun. 2004.

[6] R. Schneider, S. Krakiwsky, L. Turner, and M. Okoniewski, "Advances in Hardware Acceleration for FDTD," Chapter 20 in *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 3rd edition, Artech House, 2005.

[7] S. Adams, J. Payne, and R. Boppana, "Finite Difference Time Domain (FDTD) Simulations Using Graphics Processors," *Proceedings of the 2007 DoD High Performance Computing Modernization Program Users Group (HPCMP) Conference*, pp. 334–338, 2007.

[8] M. J. Inman, A. Z. Elsherbeni, and C. E. Smith, "GPU Programming for FDTD Calculations," *The Applied Computational*

Electromagnetics Society (ACES) Conference, 2005.

[9] M. J. Inman and A. Z. Elsherbeni, "Programming Video Cards for Computational Electromagnetics Applications," *IEEE Antennas and Propagation Magazine*, vol. 47, no. 6, pp. 71–78, December 2005.

[10] M. J. Inman and A. Z. Elsherbeni, "Acceleration of Field Computations Using Graphical Processing Units," *The Twelfth Biennial IEEE Conference on Electromagnetic Field Computation CEFC 2006*, April 30 - May 3, 2006.

[11] M. J. Inman, A. Z. Elsherbeni, J. G. Maloney, and B. N. Baker, "Practical Implementation of a CPML Absorbing Boundary for GPU Accelerated FDTD Technique," *The 23rd Annual Review of Progress in Applied Computational Electromagnetics Society*, 19-23 March 2007.

[12] M. Inman, A. Elsherbeni, J. Maloney, and B. Baker, "Practical Implementation of a CPML Absorbing Boundary for GPU Accelerated FDTD Technique," *Applied Computational Electromagnetics Society Journal*, vol. 23, no. 1, pp. 16–22, 2008.

[13] M. J. Inman and A. Z. Elsherbeni, "Optimization and parameter exploration using GPU based FDTD solvers," *IEEE MTT-S International Microwave Symposium Digest*, pp. 149–152, June 2008.

[14] M. J. Inman, A. Elsherbeni, and V. Demir, "Graphics Processing Unit Acceleration of Finite Difference Time Domain", Chapter 12 in *The Finite Difference Time Domain Method for Electromagnetics (with MATLAB Simulations)*, SciTech Publishing, 2009.

[15] N. Takada, N. Masuda, T. Tanaka, Y. Abe, and T. Ito, "A GPU Implementation of the 2-D Finite-Difference Time-Domain Code Using High Level Shader Language," *Applied Computational Electromagnetics Society Journal*, vol. 23, no. 4, pp. 309–316, 2008.

[16] A. Valcarce, G. de la Roche, and J. Zhang, "A GPU Approach to FDTD for Radio Coverage Prediction," *Proceedings of the 11th IEEE Singapore International Conference on Communication Systems (ICCS '08)*, pp. 1585–1590, November 2008.

[17] P. Sypek and M. Michal, "Optimization of an FDTD Code for Graphical Processing Units," *17th International Conference on Microwaves, Radar and Wireless Communications, MIKON 2008*, pp. 1–3, 19-21 May 2008.

[18] P. Sypek, A. Dziekonski, and M. Mrozowski, "How to Render FDTD Computations More Effective Using a Graphics Accelerator," *IEEE Transactions on Magnetics*, vol. 45, no. 3, pp. 1324–1327, 2009.

[19] N. Takada, T. Shimobaba, N. Masuda, and T. Ito, "High-speed FDTD Simulation Algorithm for GPU with Compute Unified Device Architecture," *IEEE International Symposium on Antennas & Propagation & USNC/URSI National Radio Science Meeting*, p. 4, 2009.

[20] A. Valcarce, G. De La Roche, A. Jüttner, D. López-Pérez, and J. Zhang,"Applying FDTD to the coverage prediction of WiMAX femtocells," *EURASIP Journal on Wireless Communications and Networking*, February 2009.

[21] D. K. Price, J. R. Humphrey, and E. J. Kelmelis, "GPU-based Accelerated 2D and 3D FDTD Solvers," in *Physics and Simulation of Optoelectronic Devices XV, Proceedings of SPIE*, vol. 6468, 2007.

[22] D. K. Price, J. R. Humphrey, and E. J. Kelmelis, "Accelerated Simulators for Nano-Photonic Devices," *International Conference on Numerical Simulation of Optoelectronic Devices 2007, NUSOD '07*, pp. 103–104, September 2007.

[23] A. Balevic, L. Rockstroh, A. Tausendfreund, S. Patzelt, G. Goch, and S. Simon, "Accelerating Simulations of Light Scattering Based on Finite-Difference Time-Domain Method with General Purpose GPUs," *Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering*, pp. 327–334, 2008.

[24] C. Ong, M. Weldon, D. Cyca, and M. Okoniewski, "Acceleration of Large-Scale FDTD Simulations on High Performance GPU Clusters," *2009 IEEE International Symposium on Antennas & Propagation & USNC/URSI National Radio Science Meeting*, 2009.

[25] NVIDIA CUDA ZONE: www.nvidia.com/object/cuda_home.html.

[26] Acceleware: www.acceleware.com.
[27] CUDA_Getting_Started_2.3_Windows.pdf: http://www.nvidia.com/object/cuda_develop.html.
[28] http://en.wikipedia.org/wiki/CUDA.
[29] V. Demir and A. Z. Elsherbeni, "Compute Unified Device Architecture (CUDA) Based Finite-Difference Time-Domain (FDTD) Implementation," *Journal of the Applied Computational Electromagnetics Society (ACES)*, vol. 25, no. 4, 2010.

Veysel Demir is an assistant professor at the Department of Electrical Engineering at Northern Illinois University. He received his Bachelor of Science degree in electrical engineering from Middle East Technical University, Ankara, Turkey, in 1997. He studied at Syracuse University, New York, where he received both a Master of Science and Doctor of Philosophy degrees in electrical engineering in 2002 and 2004, respectively. During his graduate studies, he worked as a research assistant for Sonnet Software, Inc., Liverpool, New York. He worked as a visiting research scholar in the Department of Electrical Engineering at the University of Mississippi from 2004 to 2007. He joined Northern Illinois University in August 2007.

Dr. Demir's main field of research is electromagnetics and microwaves. He is especially experienced in applied computational electromagnetics. He heavily participated in the development of time domain and frequency domain numerical analysis tools for new applications and contributed to research on improving the accuracy and speed of algorithms being developed. He is experienced in designing RF/microwave circuits and antennas for the related technologies, and performing experimental characterizations of these devices.

Dr. Demir is a member of IEEE and ACES and has coauthored more than 20 technical journal and conference papers. He is the coauthor of the books *Electromagnetic Scattering Using the Iterative Multiregion Technique* (Morgan & Claypool, 2007) and *The Finite Difference Time Domain Method for Electromagnetics with MATLAB Simulations* (Scitech 2009).