A Practical Look at GPU-Accelerated FDTD Performance

Mike Weldon[1], Logan Maxwell[1], Dan Cyca[1], Matt Hughes[1], Conrad Whelan[1], Michal Okoniewski[1,2]

[1] Acceleware Corp. 1600 – 37th St. SW, Calgary, AB T3C 3P1, Canada
logan.maxwell@acceleware.com, mike.weldon@acceleware.com

[2] Department of Electrical and Computer Engineering
University of Calgary, Calgary, Alberta T2N 1N4, Canada

***Abstract*—** This paper outlines several key features and conditions that impact the performance of FDTD on GPUs. It includes relevant performance measurements as well as practical suggestions on how to mitigate their impact. Among these factors are: PML depth, the number of unique materials, dispersive materials, the impact of field reads/observations, simulation orientation, and domain decomposition using multiple GPUs. The paper shows that the performance of FDTD on GPUs can be limited in certain extreme cases, but with proper care on the part of the designer these cases can be managed and maximum performance guaranteed.

***Index Terms*—** GPU, acceleration, FDTD, CPML, dispersive materials.

I. INTRODUCTION

For several years, running FDTD (Finite Difference Time Domain) [1] on graphical processing units, or GPUs, accelerators has been shown as a successful technique to reduce run times [2-4]. The fine-grained parallelism of FDTD maps well to the several hundreds of computational streaming processor cores available on modern GPU hardware. The faster memory bandwidth from GPU RAM to the GPU processing elements is also largely responsible for the observed performance gain versus traditional CPU (central processing unit) architectures.

The complexity involved when writing GPU-enabled FDTD codes involves making sure that the processing elements are not data starved. This is done through effective memory, cache and memory bandwidth management. This complexity must be addressed for every feature of FDTD, not just the basic [1] Yee updates.

Section II of this paper will introduce and explain the basics of FDTD performance on GPUs from an end user perspective. It will also detail the general limiting cases of this performance caused by the GPU architecture mentioned above.

The body of the paper, Sections III through IX, will build on this overview and introduce more advanced features and their impact on performance. These features are: PML (Perfectly Matched Layer) boundary conditions, the number of unique materials, dispersive materials, simulation orientation, observation/modification of field data during the simulation, and domain decomposition across multiple GPUs. In each case, the performance of the feature or concept is illustrated with a graph and explained in words. In addition, practical suggestions are offered for ensuring maximum performance and mitigating any adverse effects.

To illustrate these effects, Acceleware's FDTD library, version 9.x, implemented in CUDA and running NVIDIA Tesla C1060 are the software and GPU platforms of choice.

II. FUNDAMENTALS OF FDTD PERFORMANCE ON GPU

The graph below is an overview that illustrates the performance of Acceleware's FDTD library on both multi-core CPU and GPU hardware. The CPU hardware used throughout the paper is AMD Opteron™ 2214 2.2GHz, and Intel® Xeon® CPU X5550 @ 2.67GHz code-named 'Nehalem'. The GPU hardware is NVIDIA Tesla C1060 GPU

cards connected via PCI Express x16 to the host system.

Memory bandwidth is perhaps the key determinant of FDTD performance from a hardware perspective. The memory bandwidth of the C1060 architecture is 102GB/S, the Xeon X5550 architecture is 32GB/s, and the Opteron 2214 architecture is 11.8GB/s. These numbers roughly correlate with the peak performance observed for FDTD in Figure 1 below. Newer generations of both CPU and GPU hardware will continue to increase memory bandwidth.

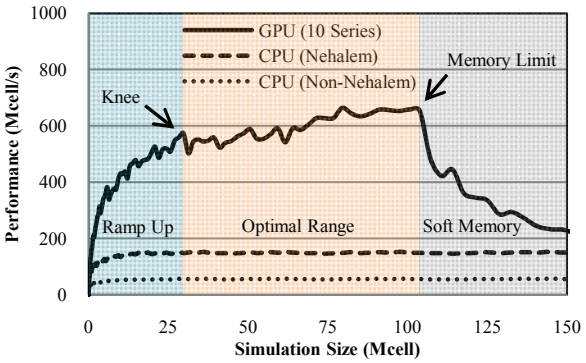


Fig. 1. FDTD performance – CPU vs. GPU.

CPU performance versus simulation size ramps up relatively quickly and reaches a constant steady state limited by the finite memory bandwidth, and provided the CPU memory is not exceeded. The GPU performance curve is more dramatic, and shows several key operating regimes which are noted in Fig. 1 and explained below.

Ramp Up - In this range the GPU is not using all of its compute resources and memory bandwidth efficiently. PML may also take up a large portion of the total simulation size and acts to slow the total simulation throughput.

Knee - The knee is the point at which the performance levels off and the GPU is running optimally.

Optimal Range - This is the optimal range for the GPU as processing resources are being fully utilized and the impact of data transfer minimized. The goal of any GPU FDTD code is to maximize the breadth and magnitude of this region.

GPU Memory Limit - This is the point at which the GPU runs out of memory. There is a clear and dramatic performance impact beyond this point due to FDTD updates shifting to the CPU. While the amount of GPU RAM is fixed, the memory limit as measured by the number of cells will change depending on the materials and features of the simulation.

'Soft Memory' - In this area the CPU is solving the remaining calculations that the GPU does not have memory for. As simulation size goes further into soft memory, the performance will approach that of the CPU.

The performance in MCells/s of throughout the paper is calculated as follows:

$$Performance\left(\frac{Mcells}{s}\right) = \frac{Simulation\ Size\ (Mcells) x\ Time\ Steps}{Simulation\ Time\ (s)} \quad (1)$$

In the above calculations, the 'simulation size' does *not* include any PML cells used in the simulation, while the 'simulation time' is the time elapsed from the beginning of the time stepping.

Unless being treated as an independent variable or otherwise noted, the performance results in this document have 16 dielectric materials, four-layer CPML (convolutional perfectly matched layer) [5], are cubic, and have field observations disabled. This includes the results in Fig. 1. While 16 materials and four-layer CPML may be a simplistic case compared with more advanced simulations which use more of each, the effect on performance is enough to be representative of a broad range of simulations. The precise dependence on more CPML and more materials are both examined in subsequent sections.

Finally, results in this paper are all reported for single precision, floating-point numerical representation of field and material data. FDTD has the advantage that the numerical dispersion is usually more significant than any single precision error. This is fortuitous since the double precision performance of GPUs has, until recently, been an order of magnitude slower than single precision.

III. PERFECTLY MATCHED LAYERS (PML)

Adding absorbing CPML (convolutional perfectly matched layer) boundary layers is done to truncate the overall simulation volume. Computation of these cells is made more intensive due to the recursive convolution performed. The number of layers required depends on the desired reflection coefficient from the boundary and is done at the discretion of the designer. Typically five to ten layers are used, with five providing -30dB or better reflection. [5]

While reducing reflections, these layers can also reduce simulation performance by as much as 50% [6], especially for small simulation volumes. The maximum simulation size the GPU is capable of running will also be partially reduced, as CPML cells require more memory than non-CPML cells. They also are more expensive to compute, which is why the performance is reduced.

Figure 2 below shows GPU-accelerated FDTD performance for simulations with different amounts of CPML. Note that both the performance and maximum GPU-accelerated simulation size are affected. Also notice that the point at which the GPU enters the soft-memory limit is reduced. This is a reflection of the increased memory usage of the CPML cells.

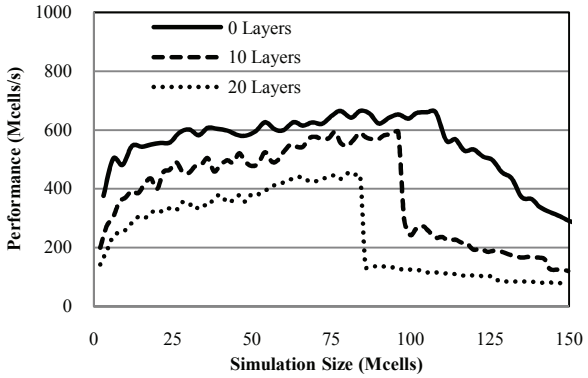


Fig. 2. GPU-accelerated FDTD performance with CPML layers.

Small simulations are impacted more than larger simulations because CPML cells represent a greater majority of the computational load.

Minimizing the use of CPML as a technique to preserve performance should be evident, and is already well known for CPU implementations. How much CPML a given simulation requires can only be understood by the designer, who will balance reflections, accuracy and performance.

IV. NUMBER OF UNIQUE MATERIALS

The number of materials, defined as materials with unique permeability, permittivity, electric and magnetic conductivity, can have a large impact on performance - up to a 20% decrease. In the case of Acceleware's implementation, this is a result of the way these properties are stored for use on the GPU.

Acceleware's library employs the well known look-up table method to save memory and avoid unnecessary copies of material data. In this technique, the look-up table index is passed to the computational kernel which in turn fetches the material parameters before completing the update. This works well (uses the least memory) for simulations where the number of unique materials is much less than the number of cells. For simulations with very large numbers of materials, it becomes more advantageous from a memory perspective to send the material parameters directly to the kernel. While there can be a performance impact to doing so, it may be more memory efficient. This so called direct technique can also support unique E and H materials up to the number of cells in the simulation.

The performance is illustrated below in Fig. 3 and shows that there is indeed a performance impact for the C1060 GPU hardware when moving from the look-up table method to the direct method. The number of unique E and H materials is also shown to have an independent and additive effect on simulation performance.

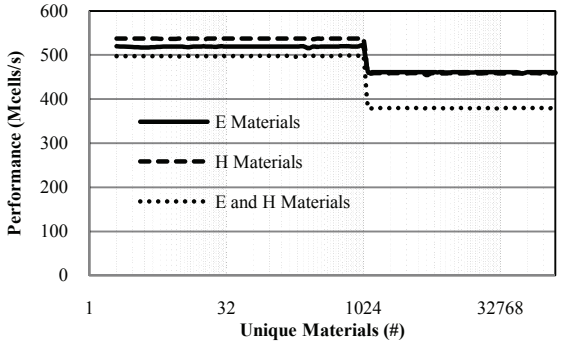


Fig. 3. GPU-accelerated FDTD performance versus the number of materials.

The choice of 1024 is somewhat arbitrary and unique to the Acceleware library. Other implementations may use a different break point or use the look-up table method or the direct method exclusively. On different GPU hardware, the direct and look-up table methods perform differently, which adds further complexity to understanding the performance.

The general end user recommendations should be to simply keep an awareness of the number of unique materials in your simulation and ensure they are not an unnecessary cause of simulation slow down. Many applications add arbitrary complexity by allowing for continuous variation of the material parameters.

For FDTD developers, additional intelligence in the library itself may also more automatically optimize the material storage and access *a priori* depending on the number of materials, hardware, and kernel implementations available. This would ensure an optimal performance where the simulation is not memory limited and maximum simulation size where it is.

V. DISPERSIVE MATERIALS

Simulating materials with dispersive properties can have an even more significant impact on simulation performance and maximum simulation size. Both the order of the dispersive materials (number of poles) and the total volume of dispersive material need to be considered. For a given cell, adding a single dispersive pole to describe its behavior will increase the memory requirement of the cell and increase the required number of flops for additional material current calculation. This increases proportionally with the number of poles. Simulations with larger volumes and more higher-order poles will hence show more pronounced degradation of performance and more reduced simulation size. Several relevant cases are shown in Fig. 4 below.

The above effect applies to all dispersive materials types: Drude, Debye, Lorentz, Drude-Lorentz, etc. Simulations with dispersive materials also run slower on the CPU, so the 'speed up factor' when using GPUs is roughly the same as for non-dispersive simulations.

Managing the effect of dispersive materials involves using only the minimum volume and order required to achieve your desired result. As is illustrated comparing cases four with five, and two with three, the distribution of dispersive materials does not significantly affect the performance; it is the overall volume and order that counts.

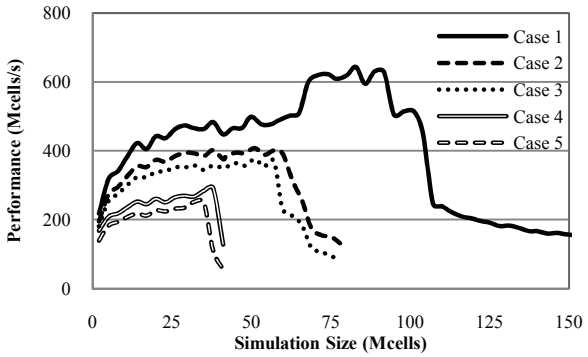


Fig. 4. GPU-accelerated FDTD performance for several cases of dispersive material usage.

Case 1 *– 1600 non-dispersive materials distributed evenly thought the entire simulation space.*

Case 2 *– 1 single-pole dispersive material occupies 40% of the total volume contiguously.*

Case 3 *– 1 single-pole dispersive distributed evenly throughout the entire volume, 40% of the total volume is made up of dispersive materials.*

Case 4 *– 1600 Multi-pole dispersive materials distributed contiguously throughout 40% of the total volume.*

Case 5 *– 1600 Multi-pole dispersive materials distributed evenly throughout the entire volume, 40% of the total volume is made up of dispersive materials.*

VI. READS AND READ REGIONS (WRITES AND WRITES REGIONS)

Moving field data between GPU and CPU system memory during a simulation can dramatically impact performance and has been discussed as a limitation of GPU FDTD implementations. For the purpose of this discussion we will refer to these data moves as *reads* and *writes*. Field data is read when calculating relevant outputs like SAR, far field patterns, optical generation, special updates for assessing convergence, and in other regards. Field data is

written in cases like soft or customized excitations or active materials.

The two critical factors affecting performance for reads and writes are: how much of the volume is read/written and how frequently. Figure 5 below shows performance for different read volumes based on a percentage of the total volume. All six fields are read for each cell in the volume. The number of time steps between each volume read is swept and shown on the horizontal axis.

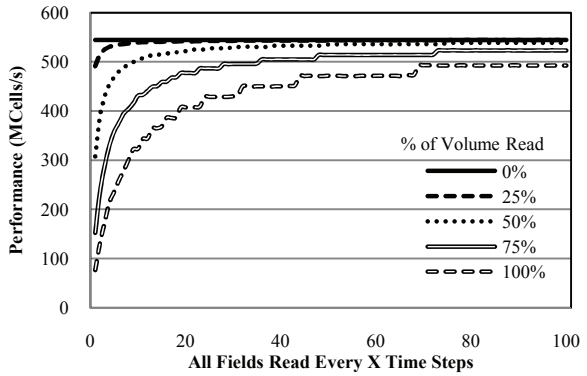


Fig. 5. GPU-accelerated FDTD performance for different field observations.

The above volume reads are made for contiguous volumes within the simulation space which is a simplified though still realistic case. The other extreme would be a large number of individual point reads dispersed evenly through a volume or plane. Individually reading these points one by one would further reduce the performance given the overhead attached with each read and their disparate locations in physical memory. Acceleware has implemented a strided region function as one technique to eliminate this overhead. An exhaustive study of all possible read patterns and techniques is a significant effort in itself and beyond the scope of this paper.

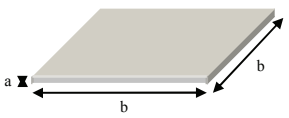
The general suggestions to manage the impact of field observations are relevant in all cases. They are: keep the read volume to a minimum, only observe the region (volume) that is of direct interest, and read only as frequently as is necessary to achieve accurate power, DFT, SAR, optical generation or other results. For steady state measurements like optical generation, far field etc. only start to read after a simulation has converged.

VII. SIMULATION ORIENTATION

Single-GPU simulations where the number of cells on one particular axis is significantly smaller than the others will experience a decrease in simulation performance and maximum simulation size. 'Significant' in this case is defined as a dimension that is 20% or less of the size of the other dimensions. For the Acceleware library, the particular dimension is Z, but this is implementation dependent.

This behavior is related to the way in which memory is optimally accessed for a given 3D layout in memory. This problem is not unique to GPU FDTD solutions; it is also present in vectorized CPU-only FDTD solvers.

The example illustrated in Fig. 6 and plotted in Fig. 7 shows an extreme case, 10:10:1, of smallest dimension. For less extreme cases the decrease in performance and max simulation size is proportionally smaller.



	X	Y	Z
X Smallest	(a,	b,	b)
Y Smallest	(b,	a,	b)
Z Smallest	(b,	b,	a)

Fig. 6. Illustration of an extreme simulation orientation case.

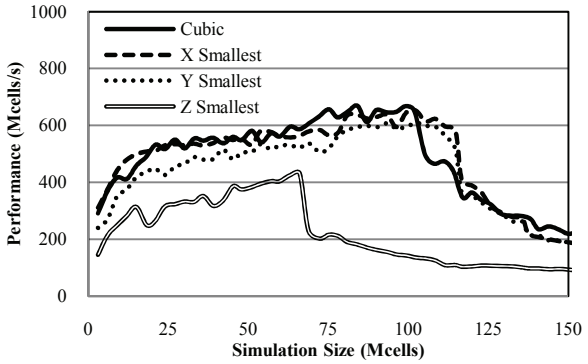


Fig. 7. GPU-accelerated FDTD performance for various extreme simulation orientations.

An important note is that partitioning across multiple GPUs will change the effective simulation dimensions on each GPU, and hence the performance, which is an important fact to consider.

To manage the effects of simulation orientation, simply rotate the simulation so that the Z (or critical) axis is not the smallest dimension. Avoid extreme differences in dimensions between the axis, if possible. Cubic simulation sizes will show the best performance.

VIII. MULTI-GPU SYSTEMS

Using multiple GPUs in concert on a single problem will increase both performance as well the maximum simulation size that can be run in full accelerated mode. Doing this requires significant additional complexity in the code, as it is not handled automatically at the hardware or driver level. The performance curve shown in Figure 1 for one GPU is now extended to show two, four and eight GPUs and plotted in Figure 8.

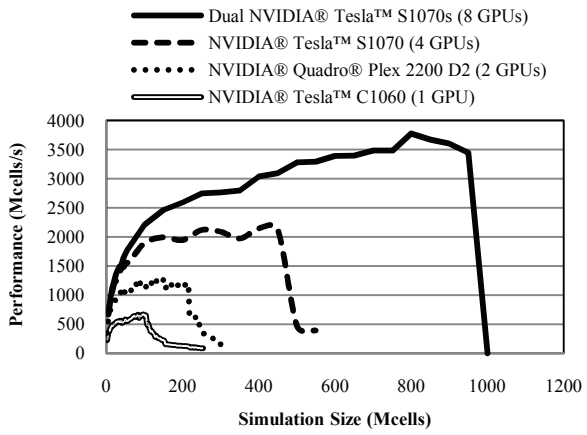


Fig. 8. GPU-Accelerated FDTD performance on multiple GPUs.

In the Acceleware implementation, the scaling with the number of GPUs depends on the simulation size. Small simulations in the ramp up range will experience a smaller scaling factor than simulations in the optimal range. For a simulation of 100 MCells, scaling is on the order of 80-90 percent up to four GPUs with diminishing returns going above four. However, if one considers the maximum throughput of each configuration, the scaling remains over 70 percent all the way up to eight GPUs. The scaling is plotted in Figure 9 below.

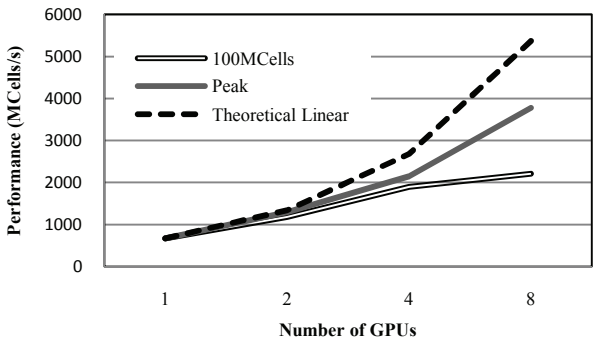


Fig. 9. GPU-Accelerated FDTD performance scaling across multiple GPUs. Peak performance observed.

Scaling beyond eight GPUs is also possible, but necessitates the use of an MPI or cluster layer above the GPU code. Clusters of up to 64 GPUs have been demonstrated at Acceleware and more details can be found in [8]. In general, GPU cluster scaling remains above 60% and well above that of CPU performance for large numbers of cores.

IX. OTHER CONSIDERATIONS

There are several other practical considerations to be aware of when running GPU accelerated FDTD. It is common to see GPUs used for FDTD computation also used to drive a display device either directly or indirectly. Display and computational work contesting for the same GPU resources can negatively impact performance. The two most common ways this can happen are with graphically intensive applications or screen savers, and with remote desktop applications.

Screen savers are not that impactful to simulation performance, typically less than 5%, but running a blank or non-3D screen saver will ensure maximum simulation performance.

Remote desktop applications on the other hand can have a severe, >50%, impact on performance and also prevent simulations from running. The best way to avoid this problem is to use a KVM, Keyboard-Video-Mouse, as it does not require any additional GPU resources. Next to that, IP-based remote desktop applications such as UltraVNC are another solution but can still reduce performance by 10-30%.

X. CONCLUSION

As GPU-accelerated FDTD has become an accepted and advantageous computational technique, its use is becoming more and more widespread. With increased usage, several practical limitations have been exposed. Most of these are in extreme cases and depend heavily on the particular implementation of accelerated FDTD functions as well as the hardware itself.

This paper looked at several of the most common practical limitations and suggested techniques to prevent them from excessively impacting performance. These included the number of materials, dispersive materials, PML absorbing boundaries, the extent of field observations, simulation orientation, and the use of multiple GPUs. Performance reductions vary from 5% to 50% versus a similar simulation in a less extreme case.

It is demonstrated that with proper care on the part of the end user, any performance degradation can be mitigated or eliminated to achieve the maximum benefit of running on GPUs. From both a SW and HW development perspective it also exposes potential architectural limitations of GPUs and should be a call to developers and designers to examine their code and hardware to further improve performance in these extreme cases.

REFERENCES

[1] K. S. Yee, "Numerical Solution of Initial Boundary Value Problems Maxwell's Equation in Isotropic Media", *IEEE Trans. Antennas and Prop*., Vol. 14, No. 3, pp. 302-307, 1966.

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

[3] P. F. Curt, J. P. Durbano, M. R. Bodnar, S. Shi, M. S. Mirotznik "Enhanced Functionality for Hardware-Based FDTD Accelerators," *ACES Journal,* Vol. 22 No. 1, 2007.

[4] P. Sypek, M. Mrozowski, "Optimization of a FDTD code for graphical processing units," *The 17th. Int. Conf. on Microwaves, Radar and Wireless Communications,* MIKON, May 2008.

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

[6] M. J. Inman, A. Z. Elsherbeni, J. G. Maloney, and B.N. Baker, "GPU Based FDTD Solver with CPML Boundaries," *IEEE Antennas and Propagation Society International Symposium*, pp. 5255- 5258, 2007.

[7] J. A. Roden and S. D. Gedney, "Convolutional PML (CPML): An efficient FDTD implementation of the CFS-ML for arbitrary media", *IEEE Transactions on Antennas and Propagation,* Vol. 50, 2002, pp. 258-265.

[8] C. Ong, M. Weldon, D. Cyca, and M. Okoniewski, "Acceleration of large-scale FDTD simulations on high performance GPU clusters," *IEEE Antennas and Propagation Society International Symposium, APSURSI '09,* pp. 1 – 4, 2009.

Mike Weldon has an MSEE from the University of Calgary / TRLabs where he researched RF to optical conversion techniques for wireless network infrastructure. He also spent five years as an RF development engineer working on broadband Doherty power amplifiers at Nortel. He is currently a product manager at Acceleware responsible for the GPU-accelerated, RF/optical FDTD product.

Logan Maxwell is presently an intern at Acceleware and will graduate next year with a BSc in Electrical Engineering from the University of Calgary.

Dan Cyca has an M. Sc. in Electrical Engineering. He has spent six years developing GPU computing products at Acceleware.

Matt Hughes is currently a senior software developer with Acceleware, working on various Professional Services projects. He was the team lead for Acceleware's FDTD product prior to heading up the linear algebra team. Prior to joining Acceleware in 2005, Matt completed a M.A.Sc in Electrical Engineering at the University of Victoria, where he studied optical transmission through thin metal films using FDTD simulations on shared memory and distributed computers. He obtained a B.Sc in Electrical Engineering from the University of Calgary in 2003.

Conrad Whelan studied Electrical and Computer Engineering at the University of Calgary where he was awarded BSc and MSc degrees. During his time with the applied electromagnetics group, he investigated conformal methods for reducing the run time of patch antenna simulations. This segued right into his work with Acceleware where he has been a member of the FDTD team for four years, seeing the full range of transition from CPU to OpenGL GPU computing to the arrival of CUDA and the integration of Multi-node MPI for cluster operations.

Michal Okoniewski, P.Eng., is a Professor at the Department of Electrical and Computer Engineering, University of Calgary. He holds Libin/Ingenuity Chair in biomdeical-engineering and Canada Research Chair in applied electromagnetics. His interests range from computational electrodynamics, to tunable reflectarrays, RF MEMS and RF micro-machined devices, as well as hardware acceleration of computational methods. He is also involved in bio-electromagnetics, where he works on tissue spectroscopy and medical imaging. Dr Okoniewski is a fellow of IEEE and member of IEEE AP-S AdCom. In 2004 he co-founded Acceleware Corp.