Progress In Electromagnetics Research, Vol. 138, 467–478, 2013

B-CALM: AN OPEN-SOURCE MULTI-GPU-BASED 3D-FDTD WITH MULTI-POLE DISPERSION FOR PLASMONICS

Pierre Wahl[1, 2, *], Dany-Sebastien Ly-Gagnon[2], Christof Debaes[1], Jürgen Van Erps[1], Nathalie Vermeulen[1], David A. B. Miller[2], and Hugo Thienpont[1]

[1]Brussels Photonics Team B-PHOT, Department of Applied Physics and Photonics, Vrije Universiteit Brussel, Belgium

[2]Department of Electrical Engineering, Stanford University, CA 940305, USA

Abstract—Numerical calculations based on finite-difference time-domain (FDTD) simulations for metallic nanostructures in a broad optical spectrum require an accurate modeling of the permittivity of dispersive materials. In this paper, we present the algorithms behind B-CALM (Belgium-CAlifornia Light Machine), an open-source 3D-FDTD solver simultaneously operating on multiple Graphical Processing Units (GPUs) and efficiently utilizing multi-pole dispersion models while hiding latency in inter-GPU memory transfers. Our architecture shows a reduction in computing times for multi-pole dispersion models and an almost linear speed-up with respect to the amount of used GPUs. We benchmark B-CALM by computing the absorption efficiency of a metallic nanosphere in a broad spectral range with a six-pole Lorentz model and compare it with Mie theory and with a widely used Central Processing Unit (CPU)-based FDTD simulator.

1. INTRODUCTION

Finite-Difference Time-Domain (FDTD) simulations play a prominent role in numerical electromagnetic calculations [1, 2]. Many problems in nanophotonics require three dimensional full-field simulations. When applying optimization schemes that require a full-field solution at each iteration step, such as genetic algorithms [3] or adjoint optimization methods [4, 5], the ability to find a solution is often limited by the speed

Received 6 March 2013, Accepted 27 March 2013, Scheduled 30 March 2013

* Corresponding author: Pierre Wahl (pwahl@b-phot.org).

of the full-field simulator. Also FDTD simulations are often limited by the available computational power. While the use of multiple Graphical Processing Units (GPUs) to accelerate FDTD simulation has been reported before [6, 7], the results have been, to our knowledge, so far limited to the implementation of flat or single-pole Drude-Lorentz dispersion material models in the microwave regime. At optical frequencies, the permittivity of metals can have more intricate features, which require including multiple resonances to obtain an accurate material model [8] that so far have only been implemented on a single GPU [9–11]. Moreover, to our knowledge, no multi-GPU-enabled FDTD simulator has been shared with the scientific community under an open source license.

The cell update in finite difference algorithms usually requires information on the state of their neighboring cells [12]. When solved on multiple GPUs this requires data transfers between the GPUs as neighboring cells can be located on different cards. Those memory transfers can potentially reduce performance due to memory bandwidth limitations or latency. This is particularly the case when the cards are located on different hosts [13]. To hide the latency, we structured B-CALM in such a way that memory transfers occur almost *asynchronously* with the execution of the simulation kernels. In this paper, we thus present an open-source GPU-based FDTD simulator that implements an algorithm to simulate multi-pole Lorentz materials on multiple GPUs while minimizing thread divergence and latency in the memory transfers between different GPUs. This enables fast simulations of complex materials. Also, as the memories of the different GPUs are aggregated, larger simulations are possible. As an example, we use B-CALM to simulate the absorption cross-section of a gold nanosphere and compare the results with Mie theory. Compared with Mie theory, we obtain an error of less than 5% on a broad spectral range and a 50-fold speedup per card compared to Meep [14], a widely used CPU-based FDTD simulator. In addition, the speed-up is almost linear with respect to the number of cards when the simulation space is large enough.

2. LORENTZ MODEL FOR DISPERSIVE MEDIA

A complex permittivity $\epsilon(\omega) = \epsilon'(\omega) + i\epsilon''(\omega)$ can be approximated over a broad range of wavelengths using the *Lorentz* multi-pole approximation. The permittivity of the dispersive material is then modeled as the sum of the spectral response of several damped

harmonic oscillators or *poles*, following expression (1) [1].

$$\epsilon_L(\omega) = \epsilon_\infty + \sum_{m=0}^{P} \frac{\omega_{pm}^2}{\omega_m^2 - \omega^2 + i\omega\Gamma_m} \tag{1}$$

As the number of poles P is increased, $\epsilon_L(\omega)$ can be approximated more accurately and over a larger bandwidth. For example, to fit the relative permittivity of gold the between wavelengths of 300 nm and 1200 nm, 6 poles are needed [15]. The exact fitting values of the pole frequency ω_m, the pole strength ω_{pm} and the pole damping Γ_m are listed in Table 1 and will be used throughout this work.

Table 1. Exact fitting values of ω_m, ω_{pm} and Γ_m for gold between the wavelengths of 300 nm and 1200 nm [15].

Parameter	$\omega_m \times 10^{16}$ (1/s)	$\omega_{pm} \times 10^{16}$ (1/s)	$\Gamma_m \times 10^{15}$ (1/s)
$m = 0$	0	1.1959	0.0805
$m = 1$	0.0630	0.2125	0.3661
$m = 2$	0.1261	0.1372	0.5241
$m = 3$	0.4510	0.3655	1.3216
$m = 5$	0.6538	1.0634	3.7887
$m = 6$	2.0235	2.8722	3.3633

3. DESCRIPTION OF THE GPU ARCHITECTURE AND KERNEL CONCURRENCY

GPUs contain several hundreds of cores that can allow for massive parallelization. The main difference compared to CPUs is that GPUs typically exploit the *Single Instruction Multiple Data (SIMD)* computer architecture [16]. This implies that all the cores assigned to an instruction set execute identical commands at each clock cycle, but on data stored at different memory addresses. The instruction set is referred to as a *kernel*. So, when a kernel is executed, all the cores that execute the kernel compute the same instruction but on different data in the GPU RAM (random access memory). The execution speed of a kernel on GPUs is often limited by *memory bandwidth, memory latency* and *divergence* [16].

Divergence occurs when, within a kernel, different threads have to perform different instructions, e.g., through an if-statement. Both paths of the if-statement are then performed *sequentially* since different instructions can not be computed at the same time under the SIMD architecture. This slows the program down significantly as the cores assigned to one path of the if-statement are busy waiting while the cores assigned to the other path are completing their instruction set [16].

The *memory bandwidth and latency* on a GPU is strongly related to the *coalescence* of the memory calls. Coalesced memory calls occur when cores request data that is located on adjacent addresses in the GPU-RAM [16]. If the data is not located on adjacent addresses, and memory calls are made to scattered locations of the GPU-RAM, the kernel is significantly slowed down, as non-coalesced memory calls are performed sequentially.

Applications using the Compute Unified Device Architecture (CUDA) manage concurrency through *streams*. A stream is a sequence of commands (possibly issued by different host threads) that is executed in a well-defined order. Different streams, on the other hand, may execute their commands out of order with respect to one another or concurrently [16]. Kernels that are allocated to different streams can therefore be executed at the same time. Streams can also be synchronized with each other.

4. MAPPING THE FDTD ALGORITHM ON MULTIPLE GPUS

The FDTD algorithm allows the calculation of propagating electromagnetic waves by alternately calculating the discretized electric and magnetic fields using a first-order spatial and temporal difference equation from Maxwell's equations [1]. For non-magnetic materials, the electric and magnetic field update equations in a cell with the permittivity described by Equation (1) are

$$\mathbf{E}^{n+1} = \underbrace{\mathbf{E}_s + C_1\mathbf{E}^n + C_2\nabla\times\mathbf{H}^{\mathbf{n+1/2}}}_{\lll\text{Update}E_A\ggg} + C^E_{\text{pml}}\Big(\underbrace{\Psi^n_{\mathbf{E}_{\perp 1}} + \Psi^n_{\mathbf{E}_{\perp 2}}}_{\lll\text{Update}E_A\ggg}\Big)$$
$$\underbrace{- C_3\mathbf{E}^{n-1} + \frac{1}{2}\sum_{p=1}^{P}\alpha_p\mathbf{J}^{\mathbf{n}}_{\mathbf{p}} + \xi_p\mathbf{J}^{\mathbf{n-1}}_{\mathbf{p}}}_{\lll\text{Update}E_B\ggg} \tag{2}$$

$$\underbrace{\mathbf{J}^{\mathbf{n+1}}_{\mathbf{p}} = \alpha_p\mathbf{J}^{\mathbf{n}}_{\mathbf{p}} + \xi_p\mathbf{J}^{\mathbf{n}}_{\mathbf{p}} + \gamma_p\frac{(\mathbf{E}^{\mathbf{n+1}} - \mathbf{E}^{\mathbf{n-1}})}{2\Delta t}}_{\lll\text{Update}E_B\ggg} \tag{3}$$

$$\mathbf{H}^{n+3/2} = \underbrace{\mathbf{H}_s + C_4\mathbf{H}^{n+1/2} + C_5\nabla\times\mathbf{E}^{\mathbf{n+1}}}_{\lll\text{UpdateH}\ggg} + C^H_{\text{pml}}\Big(\underbrace{\Psi^n_{\mathbf{H}_{\perp 1}} + \Psi^n_{\mathbf{H}_{\perp 2}}}_{\lll\text{UpdateH}\ggg}\Big) \quad (4)$$

where n denotes the time step, Δt the time increment at each time step, $\mathbf{E}$ the electric field, $\mathbf{E_s}$ the source term and $\mathbf{H}$ the magnetic field. C^E_{pml}, C^H_{pml}, are scaling constants specific to the *Perfectly Matched Layer (PML)* while $\Psi^n_{\mathbf{E}_{\perp 1}}$, $\Psi^n_{\mathbf{E}_{\perp 2}}$ $\Psi^n_{\mathbf{H}_{\perp 1}}$ and $\Psi^n_{\mathbf{H}_{\perp 2}}$ are recursive accumulators only stored in the PML regions [1]. C_1, C_2, C_3, C_4, C_5 α_p, ξ_p, γ_p are material-specific parameters and C_3, α_p, ξ_p, γ_p are only used in dispersive materials. Finally, $\mathbf{J^n_p}$ and $\mathbf{J^{n-1}_p}$ denote recursive accumulators only stored for the electric field of dispersive materials. The update Equations (2) and (3), which account for dispersion, are constructed by adding the dispersion terms that are underscored with $\lll$ UpdateE_B $\ggg$ to the regular FDTD update terms underscored by $\lll$ UpdateE_A $\ggg$ [1]. The update of the magnetic field $\mathbf{H}$ is essentially analogous to the update of $\mathbf{E}$ that is indicated as $\lll$ UpdateE_A $\ggg$.

The calculation of $\mathbf{E^n}$ and $\Psi^n_{\mathbf{E}_{\perp 1,2}}$ requires $\Psi^n_{\mathbf{H}_{\perp 1,2}}$ and $\mathbf{H}$ fields of neighboring cells as illustrated in Fig. 1(a). In contrast, as is clear from Equation (3), the calculation of $\mathbf{J^{n+1}_p}$ requires $\mathbf{J^n_p}$, $\mathbf{J^{n-1}_p}$, $\mathbf{E}^n$ and $\mathbf{E^{n-1}}$, which are associated with the calculated cell only. Therefore, their update is entirely local as no fields associated with neighboring cells are needed, as illustrated in Fig. 1(b).

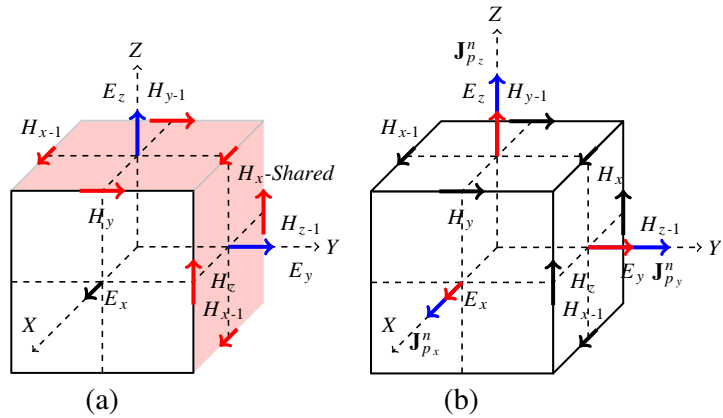

Figure 1. Data needed for the update of a dispersive cell. (a) The field update of the electric field $\mathbf{E}$ and $\Psi^{n+1}_{E_{\perp 1,2}}$ (in blue) requires the neighboring $\mathbf{H}$ fields (in red) to be loaded. Some of those fields are shared. (b) The update of $\mathbf{J}^n_p$ (in blue) requires no knowledge of the neighboring fields, but only of the electric field in the same direction (in red).

4.1. An Additional Kernel to Alleviate Thread Divergence

In previous GPU implementations of FDTD [17], the update equations are split into two kernels, one for the electric and one for the magnetic field update equation as illustrated in Fig. 2(a). However, as $\mathbf{J_p^n}$ only needs to be updated for dispersive materials, the electric field update creates a slow diverging path, since, in general, not all cells are dispersive. The situation is exacerbated for materials with a higher number of poles, as the calculation is done sequentially for the diverging paths. As described in references [9–11], B-CALM separates the electric field update equation into two separate kernels (labeled $\lll$ UpdateE_A $\ggg$ and $\lll$ UpdateE_B $\ggg$ in Fig. 2(b)) to alleviate this issue. The cost per field is one extra read and write operation to the device memory to store the intermediate result as both kernels are called sequentially. As $\lll$ UpdateE_B $\ggg$ only uses fields of one cell, it is only responsible for $2(P+1)$ read-write operations per field† making the split kernel approach faster as soon as the number of poles $P > 1$.

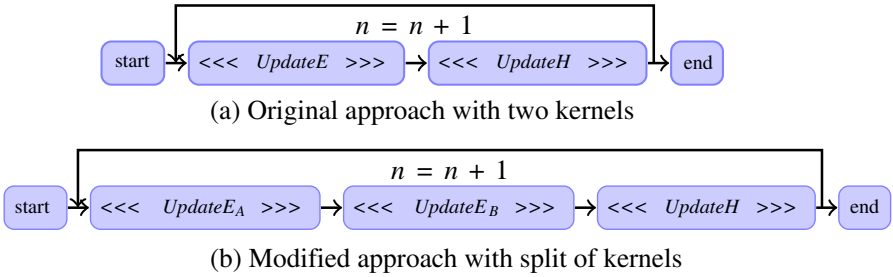

Figure 2. Comparison of the original approach (a) with the split kernel approach where computation and communication overlap (b).

4.2. Memory Scheme

To allow for multi-GPU implementation, the simulation space is split in equal parts along the Z direction and each part is allocated to a specific GPU as depicted in Fig. 3(a). On each GPU, the slower and large *device memory* is only used to store the fields **E**, **H** and the recursive accumulators for the C_{PML}s and the dispersive material parameters. To minimize memory transfers, the material parameters C_1, C_2, C_3, C_4, C_5, α_p, ξ_p, that remain constant throughout the simulation, are stored in a fast read-only *texture memory*. Also, fast shared memory is used as described in reference [12], so that the electromagnetic fields

† This corresponds to $\mathbf{E}^n$, $\mathbf{E^{n-1}}$ for each field and $\mathbf{J_p^n}$, $\mathbf{J_p^{n-1}}$ per pole per field.

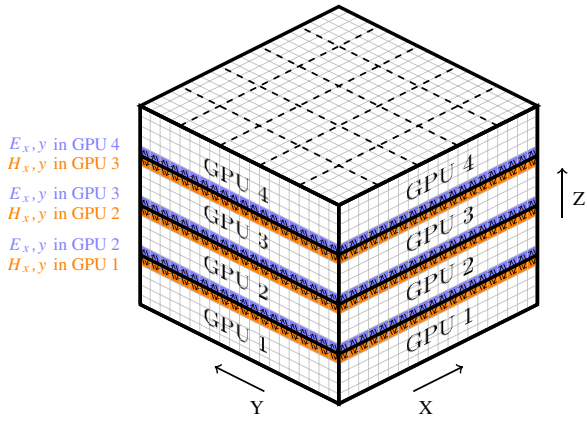

Figure 3. Mapping of the FDTD simulation space on different GPUs.

in neighboring cells are only loaded once per update and that those loads are coalesced. In this way, multiple fields can be loaded during the same cycle, which allows for a more efficient use of the memory bandwidth.

As depicted in Fig. 3 the bottom layer of electric fields $E_{x,y}$ in each GPU needs the magnetic fields $H_{x,y}$ in the top layer of the previous GPU. Conversely, for the update of $H_{x,y}$ in the top layer of each GPU, the bottom $E_{x,y}$ in the previous GPU is required. There are two possible ways to organize these transfers. The simplest way is to make the transfers *synchronously* with the rest of the simulation as it is done [6]. After the execution of $\lll$ Update$H$ $\ggg$ the updated H fields on the boundary between two GPUs are copied to the next GPU so that it can be used by $\lll$ UpdateE_A $\ggg$ and $\lll$ UpdateE_B $\ggg$ to calculate the electrical fields. Finally the updated $E_{x,y}$ on the lower borders are transferred back so they can be used in the next update of $\lll$ Update$H$ $\ggg$. Both simulation and transfer kernels are associated with one dedicated stream per card. These streams are synchronized with each other after every transfer step. The main advantage of this technique is its simplicity. However its disadvantage is that the simulation kernels are idle while the transfers between the cards are performed. This idle time can be significant depending on the simulation size and on how the cards communicate. This situation is exacerbated when the GPUs are located on different hosts. It is possible to hide this latency by using a slightly more complicated *asynchronous* transfer mechanism [13]. The main difference with the synchronous mechanism is that the cells that need to be transferred to another GPU are computed and transferred asynchronously with the cells that do not. This is illustrated in Fig. 4(b). Two streams per

card are defined: stream 1 is associated with the update and transfer of the border cells and stream 2 is associated with the update of the remainder of the simulation space. All the streams are synchronized with each other after each transfer. As can be seen from Fig. 4(b) the transfer time between GPUs can be hidden by the main simulation computed in stream 2.

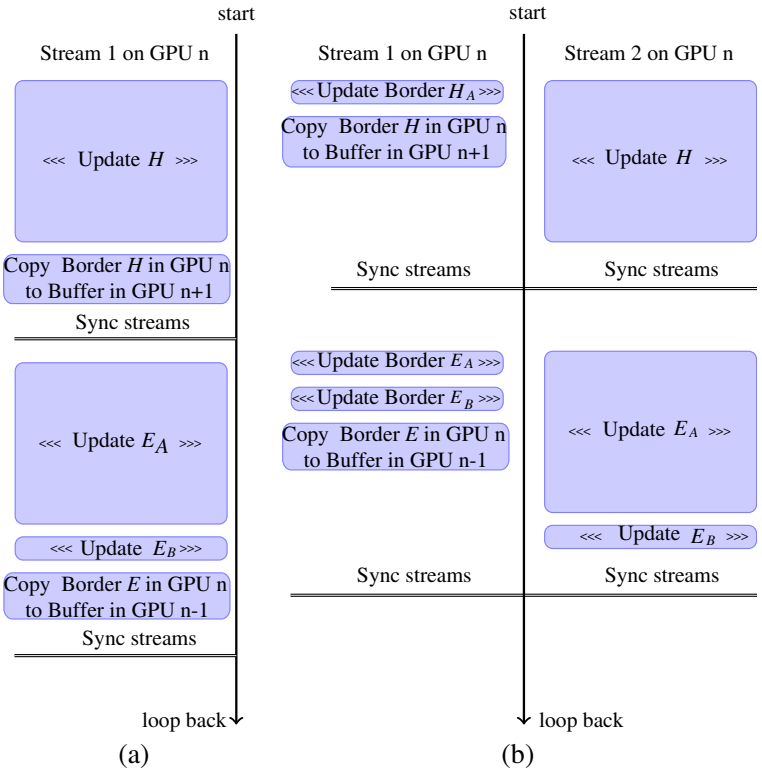

Figure 4. Schemes for the FDTD-implementation on multiple GPUs with (a) synchronous and (b) asynchronous data transfers.

5. RESULTS

We use B-CALM to calculate the absorption efficiency of a metallic sphere of 80 nm diameter under plane wave illumination and compare it with Mie theory [18] using the modeled values of the permittivity. We use a uniform mesh of 0.5 nm and $208 \times 208 \times 400$ cells with two symmetry planes and a 15-cell-wide PML. We calculated the absorption efficiency of the sphere by first integrating the Poynting vector on each face of a closed box surrounding the sphere. Then

we subtracted the output flux from the input flux through the box surrounding the sphere. As shown in Fig. 5, the absorption efficiency obtained using B-CALM closely matches the analytical value obtained by Mie theory, with a relative error smaller than 5% over the entire simulated spectrum.

The simulation speed is 1.82×10^{10} cells/min for a six-pole dispersion model on a single NVIDIA C-2075 GPU, which is 50 times faster than with Meep [14] on a Quad Core Intel(R) Xeon(R) CPU X5650 processor.

In order to test the speed as a function of the grid size, we perform the same simulation but we vary the mesh size and consequently the amount of cells in the grid. These simulations are run on 1, 2, 3 and 4 cards at the same time to test scalability. The results are compiled in Fig. 6 where the speed of Meep and B-CALM is compared for different

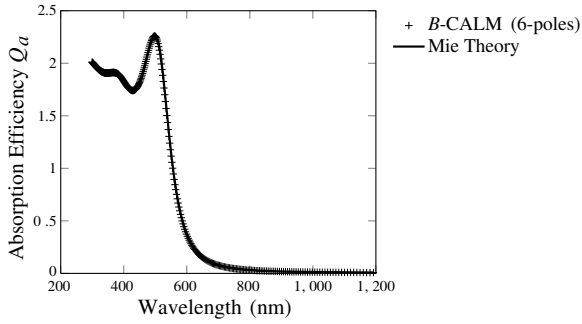

Figure 5. Simulated absorption efficiency of a gold nanosphere with a diameter of 80 nm. The simulated absorption cross-section closely matches the theoretical value over a broad wavelength range.

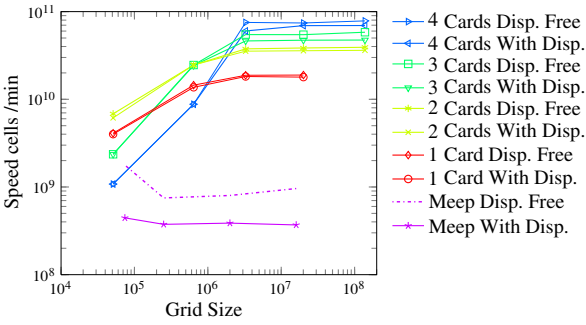

Figure 6. Simulation speed of B-CALM running on different numbers of GPUs, with or without taking dispersion into account, for a metallic nanosphere on different grid sizes. A comparison with the CPU-based FDTD simulator Meep [14] is also included.

simulation sizes and for a different number of cards with and without dispersion. When we use 4 cards this scales up to 6.94×10^{10} cells/min which is 3.8 times faster than the simulation on one card and 187 times faster than Meep. Note that while the introduction of dispersion in Meep significantly slows down the simulation speed, it only slightly reduces the simulation speed of B-CALM. We attribute this to the introduction of a dedicated kernel for dispersive cells. Also, the speedup is somewhat reduced for smaller grid sizes. We believe that this is due to the parallelization overhead becoming dominant and because the simulation is too small to efficiently hide memory transfers inside the GPU.

6. CONCLUSIONS

We have shown that dispersive materials with complex wavelength dependence can be accurately simulated using a multiple-GPU-based FDTD while preserving the speed and the low-cost advantage of GPUs. B-CALM, our GPU-accelerated open-source 3D-FDTD simulator, allows us to quickly simulate complex metal structures over a broad wavelength range and on multiple GPUs while hiding the latency involved in the memory transfers. B-CALM currently supports any user-defined sources, dedicated 2D kernels, variable grid size and PML, and is structured to easily allow the implementation of non-linearity and anisotropy. B-CALM and its user-friendly interface with Matlab can be freely downloaded at http://b-calm.sf.net.

ACKNOWLEDGMENT

The authors acknowledge the support of the Belgian American Education Foundation, the Methusalem Foundation, the ZAP, VUB OZR and the Interconnect Focus Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity. The authors acknowledge Jurgen Van Erps and Nathalie Vermeulen for their valuable comments and suggestions.

REFERENCES

1. Taflove, A., S. C. Hagness, et al., *Computational Electrodynamics: The Finite-difference Time-domain Method*, Artech House, Norwood, MA, 1995.

2. Junkin, G., "Conformal FDTD modeling of imperfect conductors at millimeter wave bands," *IEEE Transactions on Antennas and Propagation*, Vol. 59, No. 1, 199–205, Jan. 2011.
3. Gondarenko, A. and M. Lipson, "Low modal volume dipole-like dielectric slab resonator," *Opt. Express*, Vol. 16, No. 22, 17689–17694, 2008.
4. Jensen, J. S. and O. Sigmund, "Topology optimization of photonic crystal structures: A high-bandwidth low-loss T-junction waveguide," *JOSA B*, Vol. 22, No. 6, 1191–1198, 2005.
5. Hansen, P., Y. Zheng, E. Perederey, and L. Hesselink, "Nanophotonic device optimization with adjoint FDTD," *CLEO: Applications and Technology, Optical Society of America*, 2011.
6. Nagaoka, T. and S. Watanabe, "Multi-GPU accelerated three-dimensional FDTD method for electromagnetic simulation," *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC*, 401–404, IEEE, 2011.
7. Stefanski, T. P., N. Chavannes, and N. Kuster, "Multi-GPU accelerated finite-difference time-domain solver in open computing language," *PIERS Online*, Vol. 7, No. 1, 71–74, 2011.
8. Palik, E. D. and G. Ghosh, *Handbook of Optical Constants of Solids*, Academic Press, 1998.
9. Wahl, P., D. S. Ly-Gagnon, C. Debaes, D. A. B. Miller, and H. Thienpont, "B-calm: An open-source GPU-based 3D-FDTD with multi-pole dispersion for plasmonics," *2011 11th International Conference on Numerical Simulation B-CALM: An Open-Source multi-GPU-based 3D-FDTD with Multi-Pole Dispersion for Plasmonics13 of Optoelectronic Devices (NUSOD)*, 11–12, IEEE, 2011.
10. Shahmansouri, A. and B. Rashidian, "GPU implementation of split-field finite-difference time-domain method for drude-lorentz dispersive media," *Progress In Electromagnetics Research*, Vol. 125, 55–77, 2012.
11. Lee, K. H., I. Ahmed, R. S. Goh, E. H. Khoo, E. P. Li, and T. G. Hung, "Implementation of the FDTD method based on lorentz-drude dispersive model on GPU for plasmonics applications," *Progress In Electromagnetics Research*, Vol. 116, 441–456, 2011.
12. Micikevicius, P., "3D finite difference computation on GPUs using CUDA," *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, ACM*, 79–84, 2009.
13. Playne, D. P. and K. A. Hawick, "Comparison of GPU archi-

tectures for asynchronous communication with finite-differencing applications," *Concurrency and Computation: Practice and Experience*, 2012.

14. Oskooi, A. F., D. Roundy, M. Ibanescu, P. Bermel, and S. G. Johnson, "Meep: A flexible free-software package for electromagnetic simulations by the FDTD method," *Computer Physics Communications*, Vol. 181, No. 3, 687–702, Mar. 2010.
15. Rakic, A. D., A. B. Djurišic, J. M. Elazar, and M. L. Majewski, "Optical properties of metallic films for vertical-cavity optoelectronic devices," *Applied Optics*, Vol. 37, No. 22, 5271–5283, 1998.
16. Nvidia, "Nvidia cuda programming guide," NVIDIA, 2011.
17. Zunoubi, M. R. and J. Payne, "Analysis of 3-dimensional electromagnetic fields in dispersive media using cuda," *Progress In Electromagnetics Research*, Vol. 16, 185–196, 2010.
18. Ishimaru, A., *Wave Propagation and Scattering in Random Media*, Wiley-IEEE Press, 1999.