Progress In Electromagnetics Research M, Vol. 17, 135–150, 2011

GPU APPROACH FOR HERTZIAN POTENTIAL FORMULATION TOOL ORIENTED FOR ELECTROMAGNETIC NANODEVICES

D. Tartarini

Scuola Superiore ISUFI
University of Salento, via Arnesano 16, Lecce 73100, Italy

A. Massaro

Center of Bio-Molecular Nanotechnology
Italian Institute of Technology IIT
Via Barsanti, Arnesano, Lecce 73010, Italy

Abstract—The time domain modeling and simulation of electromagnetic (EM) waves interaction with nanodevices, at high spatial and time resolution, requires high computational power. For the first time, in this paper we present an effective implementation of the Hertzian Potential Formulation (HPF) on the Graphics Processing Units (GPUs), through the NVIDIA's CUDA (Compute Unified Device Architecture) programming model. It accelerates the nanodevice EM simulations at nanometer scale harnessing the massive parallelism of the GPU based systems. This study is useful for similar electromagnetic codes including the Finite Difference approaches. The results demonstrate that this GPU tool outperforms the CPU based HPF implementation, reaching a speedup from $30\times$ to $70\times$.

1. INTRODUCTION

Having a high resolution in space and in time is fundamental to generating numerically accurate simulations of electromagnetic (EM) wave interactions in nanodevices [1]. It is quite common to design dielectric devices embodying nanoparticles, quantum dots, nanorods or other nano discontinuities. The interest is in simulating at nanometer resolution the behavior of these devices with an overall size of tens or hundreds of nanometers. In particular the critical configurations

Received 3 February 2011, Accepted 8 March 2011, Scheduled 14 March 2011
Corresponding author: Daniele Tartarini (daniele.tartarini@gmail.com).

where the accuracy could fail at optical frequencies are: nanodefects, high bulk dimension in bulk-type structures, nanostructures in long optical fiber, singularity points, very thin dielectric layer in very long dielectric structure, dielectric discontinuities.

Traditional numerical methods such as the Finite Element Method (FEM), the Beam Propagation Method (BPM) [2], and the widely used FDTD (Finite Difference Time Domain) by Yee [3], require huge computational power to achieve the required level of accuracy of the EM solution.

Recently a new method, the HPF (Hertzian Potential Formulation) [1, 4, 5], has been proposed for the simulation of full-wave propagation and reflection in the time domain. The main advantage of the HPF method is that it is computationally more efficient than the Yee FDTD [3]. In the one-dimensional domain it performs half of the computation, needing to solve only two instead of the four finite-difference equations of the Yee's algorithm in the two-dimensional case [1]. Notwithstanding, accurate nanoscale simulations require great computing power that can be found not only in the expensive HPC (High Performance Computing) systems based on CPUs but also in the GPU (Graphics Processing Units) based solutions.

In fact, a recent extraordinary evolution, from simple hardwired Video Graphic Array (VGA) controllers to highly multithreaded multiprocessors optimized for visual computing, made the GPUs mature massive parallel multiprocessors. In particular, they adopted wider bandwidth and a more general-purpose architecture that allows their usage as HPC systems with very low power consumption [8]. Moreover, given the wide integration in commercial off the shelf computers, they represent a powerful and cheap solution to computational science. GPU computing is being exploited in many scientific applications [6–8] with interesting results in the EM field and nanotechnology [9, 10]. Much published work on EM computational simulations using the GPUs confirms the growing interest in this powerful technology of the computational EM community [11–13].

In this work we developed on GPUs a HPF based tool to accelerate the simulation of EM wave interaction with structures at nanoscale resolution.

The proposed HPF implementation is important in order to predict the real behavior of a nanodevice and can be applied for the EM characterization of nanosensors. The time-domain HPF algorithm has been fully implemented on GPU, through a new computational pattern, to exploit their high computational power.

Numerical results demonstrate that the proposed GPU-based HPF can significantly improve the computational efficiency. The

core of the HPF algorithm presents a data parallelism that could harness the massive parallelism of the many-cores structure of the GPUs. In fact the parallel version of HPF, implemented in this work through the NVIDIA CUDA (Compute Unified Device Architecture) architecture, achieves an overall speedup factor ranging from $30\times$ to $70\times$, depending on the geometrical configuration and single/double precision used.

In the remainder, this paper is organized as follows: Section 2 gives a brief description of the HPF method and the serial algorithm; Section 3 describes the GPU CUDA hardware architecture, the parallelization strategy of the CUDA implementation of HPF and the computational pattern; Section 4 describes the numerical results and finally Section 5 closes with the conclusions.

2. THE HPF METHOD AND THE THEORY FORMULATION

In the new HPF all the possible electromagnetic field components, which can describe the scattering and radiation effect of a nanodefect, are defined by the Hertzian electric and magnetic vectors as:

$$\begin{aligned}\overline{\mathbf{E}} &= \nabla\nabla\cdot\overline{\Pi}_e - \varepsilon\mu\frac{\partial^2}{\partial t^2}\overline{\Pi}_e - \mu\frac{\partial}{\partial t}(\nabla\times\overline{\Pi}_h)\\ \overline{\mathbf{H}} &= \nabla\nabla\cdot\overline{\Pi}_h - \varepsilon\mu\frac{\partial^2}{\partial t^2}\overline{\Pi}_h - \varepsilon\frac{\partial}{\partial t}(\nabla\times\overline{\Pi}_e)\end{aligned} \tag{1}$$

The potential $\overline{\Pi}_e$ is often called the electric Hertzian potential since it is generated from electric currents and satisfies the same boundary conditions as the electric field. In the same manner $\overline{\Pi}_h$ is often referred to as the magnetic Hertzian potential. By using the Hertzian scalar potentials ψ^e and ψ^h, the electromagnetic fields are then determined by the following expressions:

$$\begin{aligned}\overline{\mathbf{E}}=&\hat{x}A^e+\hat{y}B^e+\hat{z}C^e-\hat{x}\mu\varepsilon\partial_t^2\psi^e-\hat{y}\mu\varepsilon\partial_t^2\psi^e-\hat{z}\mu\varepsilon\partial_t^2\psi^e\\ &-\mu\partial_t(\hat{x}(\partial_y\psi^h-\partial_z\psi^h)+\hat{y}(\partial_z\psi^h-\partial_x\psi^h)+\hat{z}(\partial_x\psi^h-\partial_y\psi^h))\\ \overline{\mathbf{H}}=&\hat{x}A^h+\hat{y}B^h+\hat{z}C^h-\hat{x}\mu\varepsilon\partial_t^2\psi^h-\hat{y}\mu\varepsilon\partial_t^2\psi^h-\hat{z}\mu\varepsilon\partial_t^2\psi^h\\ &+\mu\partial_t(\hat{x}(\partial_y\psi^e-\partial_z\psi^e)+\hat{y}(\partial_z\psi^e-\partial_x\psi^e)+\hat{z}(\partial_x\psi^e-\partial_y\psi^e))\end{aligned} \tag{2}$$

$$\begin{aligned}A^{e,h}=&\partial_x^2\psi^{e,h}+\partial_{yx}\psi^{e,h}+\partial_{zx}\psi^{e,h}\\ B^{e,h}=&\partial_{xy}\psi^{e,h}+\partial_y^2\psi^{e,h}+\partial_{zy}\psi^{e,h}\\ C^{e,h}=&\partial_{xy}\psi^{e,h}+\partial_{yz}\psi^{e,h}+\partial_z^2\psi^{e,h}\end{aligned} \tag{3}$$

The HPF model proposed is iterative, i.e., at each iteration time step the electromagnetic field information of each point of the structure must be properly updated.

HPF Sequential Algorithm

(1) Setting permittivity mask parameters $\alpha(j,l), \beta(j,l), \gamma(j,l), \sigma(j,l)$ for device and nanostructures[†].
(2) Initialize data for Mur ABC conditions
(3) **for** $t = 0 \rightarrow$ max_timestep
(4) **for** $l = 0 \rightarrow$ length
(5) **for** $j = 0 \rightarrow$ width
(6) compute source values and ABC Mur conditions [14].
(7) update EM field: $\psi^{e,h}(j,l,t) = \alpha(j,l)\psi^{e,h}(j,l,t-1) + \beta(j,l)\psi^{e,h}(j,l,t-2) + \gamma(j,l)\psi^{e,h}(j,l+1,t-1) + \sigma(j,l)\psi^{e,h}(j,l-1,t-1)$
(8) **end for**
(9) **end for**
(10) **end for**.

The potentials $\psi^{e,h}(x,y,z,t)$ represent the solutions of the homogeneous wave equations for a non-dissipative medium:

$$\nabla^2\psi_{e,h}(x,y,z,t) - \mu\varepsilon\frac{\partial^2\psi_{e,h}(x,y,z,t)}{\partial t^2} = 0 \tag{4}$$

having as solution in the 2D space the following iterative form:

$$\psi^{t+1}(j) = \psi^t(j+1)\left(\frac{b}{a}\right) + \psi^t(j)\left(\frac{2a-2b}{a}\right) + \psi^{t-1}(j)(-1) + \psi^t(j-1)\left(\frac{b}{a}\right) \tag{5}$$

and for a dissipative medium:

$$\nabla^2\psi_{e,h}(x,y,z,t) - \mu\varepsilon\frac{\partial^2\psi_{e,h}(x,y,z,t)}{\partial t^2} - \mu\sigma\frac{\partial\psi_{e,h}(x,y,z,t)}{\partial t} = 0 \tag{6}$$

having as solution in the 2D space the following iterative form:

$$\psi^{t+1}(j) = \psi^t(j+1)\left(\frac{b}{a'}\right) + \psi^t(j)\left(\frac{2a'-2b}{a'}\right) + \psi^{t-1}(j)(-1) + \psi^t(j-1)\left(\frac{b}{a'}\right) \tag{7}$$

where $\Delta\varepsilon$ represents the variation of permittivity ε [4] (ε is the electrical permittivity expressed in farads/meter), μ is magnetic permeability (henrys/meter), and σ is the electric conductivity (siemens/meter) which is zero in a perfect dielectric.

[†] Constants correspond to those in Eqs. (5) (7) where: $\alpha(j,l) = (2a-2b)/a\beta(j,l) = -1\gamma(j,l) = \sigma(j,l) = b/a$. [14, 15].

The proposed HPF formulation is suitable for the following physical problems:

-Maxwell equations solution on dielectric discontinuities: this kind of problem usually is solved by numerical and analytical approximations (such as effective dielectric constant (EDC) method [1]). By means of the EDC approach (analytical approximation) the dielectric multilayer structure is considered as a single medium with an effective refractive index. In this case the analytical approximation does not consider the imperfection effects of the layer roughness (scattering, reflection, etc.) which are very important in a complex 3D structure with irregular dielectric profile.

-Bulk dimension in bulk-type structures: common integrated structures are growth on a substrate material which is characterized by a large thickness compared with the thickness of the guiding dielectric layer. Usually an EM simulator takes into account a small bulk thickness not considering the real power losses of the bulk material. The HPF combined with analytical approaches allows to overcome the problem of simulation associated with the dimension of the bulk [1].

-Nanostructures in long optical fiber: nanostructures such as photonic crystal can be realized in optical fiber. Photonic crystal fiber are very long (of the order of the meter), and so only lengths of few hundred of microns (high aspect ratio) can be simulated not considering the distributed effect on the total length and the radius of curvature effect (field losses) as in the real case of the experimental setup.

-Singularity points: in order to evaluate the near field around a singularity point such as dielectric corners, it is important in this case to discretize the volume which embeds the corners in very fine mesh size. A non uniform mesh (sub-griding) can solve the problem related to the accuracy of the solution near a singularity region, but for a large number of 3D singularity points arranged in complex 3D structure the computational cost becomes very high.

-Nanodefects: nanodefects in optical waveguide are actually used in many optical applications in which the combined effect of nanodefects is required in order to enhance the optical intensity. Several arrangements of nanostructures in semiconductor, such as quantum dots and point defects in crystals, characterize the light coupling of optical waveguides. The coupling behavior of these imperfections can be analyzed with accuracy only for a small number of defects. The combined coupling effect of more defects can be simulated by means of a proper GPU implementation.

The numerical tool presented in this work, based on HPF [1, 4, 5], overcomes these limits (big RAM memory, and high aspect ratio) by

combining the accuracy of the method with the GPU implementation. The HPF approach is intrinsically well suited to the applications of the concept of circuit analysis and synthesis, and represents a novel alternative method in the analysis of nanodefects in optical waveguide, by providing a good convergent numerical solution. This circuital analogy allows to model nanodefects in dielectric material as a set of transmission line circuits which take into account the dielectric interfaces along the propagating direction as voltage and current generators [1, 4, 5]. We show as an example in this work the wavefront propagation in a very long material characterized by nanodefects. The model highlights the accuracy of method for different patterns of nanodefects.

3. THE PARALLELIZATION STRATEGY FOR HPF WITH CUDA PROGRAMMING MODEL

3.1. The GPU Architecture and CUDA Programming Model

To completely appreciate the performance and the power of the GPU computing, it is useful to properly comprehend how the graphics processors provide the massive parallelism and why the CUDA programming model is so effective. From the hardware point of view, a GPU system, since the Tesla architecture, is a multiprocessor made of an array of tens of multiprocessors, called *Streaming Multiprocessors* (SMs) [16]. Basically, each SM is built up of two *Special Function Unit (SFU)* and eight scalar processor cores, called *Streaming Processor*s (SPs). These SMs provide each one of their SPs with one 32-bit *register file* and low latency on-chip *shared memory*. Since this architecture embeds also multithreading and scheduling functionality in hardware, thousands of threads run on hundreds of cores very efficiently, in a scalable and transparent way.

To harness this multiprocessors massive parallelism, NVIDIA conceived the CUDA [17], a software platform with a new powerful API, which extends the C/C++ language with a scalable parallel programming model. It has an SPMD (Single Program Multiple Data) [19] software style that allows writing serial code for one thread, whilst it is instanced and executed by thousands of threads in parallel. Furthermore the programmer arranges the threads in a two level architecture provided by CUDA: three-dimensional *blocks* of threads organized into a two-dimensional *grid* of blocks. This architecture drives the partition of the problem into sub-problems, which can be solved concurrently by independent blocks of threads, and into smaller pieces that can be computed in parallel by each thread. The

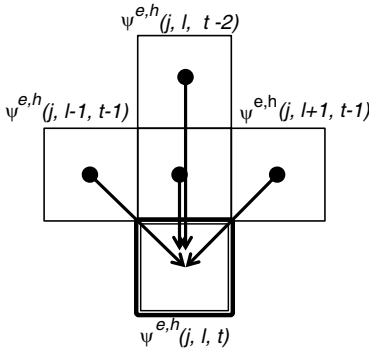

Figure 1. HPF algorithm stencil showing data dependency for the $\psi^{e,h}$ update.

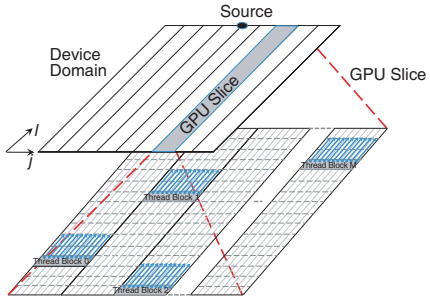

Figure 2. Domain decomposition scheme: device domain is divided in slices provided to GPU's kernel. The data domain is further partitioned in blocks whose threads run HPF concurrently.

programmer codes in the application the size of the blocks by taking into account that all the threads of a block will reside, with their own memory context, in a SM's registers and memories [18]. Furthermore a smart choice of blocks size and a proper usage of the memory hierarchy can improve performance, as we will show in the Numerical results section. A good introduction of the thread model and hardware implementation of CUDA can be found in [16–20].

In this work we tested our implementation of the tool on a GPU system of the Tesla generation: An NVIDIA GTX295 with 30 SMs for each of its 2 GPUs. It is worth noting that independently of the underlying hardware, a CUDA application is able to run transparently on every GPU.

3.2. The Parallelization Strategy and Domain Decomposition for HPF

The simulation of the EM device behavior requires the domain discretization into a two-dimensional grid of size $n_x \times n_y$ points (represented by cells), which depends on the geometry and on the spatial resolution needed. In fact, for a given device size, the better is the resolution the greater is the amount of the grid points. As shown in Fig. 2, the EM source is at the top edge of the device and the wavefront propagates along the l-direction, according to the assigned coordinate system. From a computational point of view, the heaviest part of the

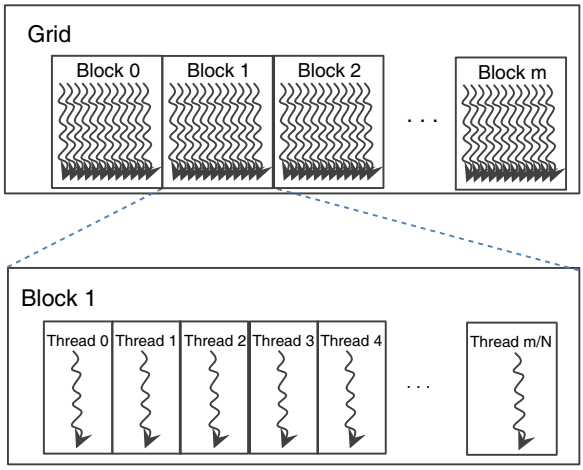

Figure 3. The CUDA organization of the threads in grid and blocks. In HPF a thread updates all the points along a given j-coordinate for each time steps. Therefore they are deployed along a linear (one-dimensional) grid of blocks.

HPF algorithm, which benefits from the parallel computing on GPU, is the updating of the EM field, (see the row (7) in algorithm reported in HPF sequential algorithm in Section 2). Therefore our parallelization strategy assigns to each thread the updating in time of the EM field of the grid points. It takes into account the constraints of HPF algorithm, in time and space, given by the data dependency scheme shown in Fig. 1 and Fig. 6. Each update of the $\psi^{e,h}(j,l,t)$, uses the values of the previous two time steps, $\psi^{e,h}(j,l,t-1), \psi^{e,h}(j,l,t-2)$, and the points one position back and forward in the wave-front propagation direction, $\psi^{e,h}(j,l-1,t-1), \psi^{e,h}(j,l+1,t-1)$. Therefore the points belonging to a line parallel to the wave-front direction are updated consecutively, along the ascending l-coordinate. Moreover these lines are independent of each other and are updated in parallel by concurrent threads, one thread per line (Fig. 2).

To exploit this parallelism, in our CUDA implementation the kernel is executed by a one-dimensional grid of one-dimensional blocks of threads, (Fig. 3). Therefore each thread of a thread block has the task to update only the points along a j-coordinate.

Furthermore, in order to ensure scalability and to deal with the memory resource constraints, the device domain is partitioned into slices (called GPU slices) that, in their turn, are assigned to a CUDA

grid of thread blocks (Fig. 2). The evolution in time of the $\psi^{e,h}$ is thus computed by thousands of independent threads that load and store data from the GPU DRAM global memory.

Even though GPU has huge computational power and a wide bandwidth toward its DRAM memory, the performances are often affected by the big latency of global memory access and by the bandwidth [21].

In the case of the HPF algorithm (see row (7)), each update of $\psi^{e,h}$ in a point requires four read accesses to $\psi^{e,h}$ values in global memory, four accesses to read the permittivity mask values and one access to store the result. The number of clock cycles wasted in waiting data from global memory is at least one order of magnitude greater than those spent in computing [20]. Therefore, we propose a solution based on the principle of locality [21] that improves the computational efficiency lowering the overall computational time. This solution reduces the number of global memory accesses by keeping data to be reused into the shared memory (global memory accesses have a latency 100 times greater than the latency of the accesses to the shared memory).

A thread can update the points shown in Fig. 4, preserving the data dependency of the HPF, in two different ways: lexicographically or along diagonals. The latter option is preferred because it does not require ghost points, synchronization or extra computation when the domain is long and further divided. These diagonals are considered consecutively from left to right, while their points are updated from the top to the bottom. In Fig. 4 is shown one of these diagonals, highlighted with dark grey background, which starts in coordinate $l = L$ and $t = 2$. To update its points, the thread accesses $2d + 2$ times the global memory to read the neighbour points required by HPF. They belong to the diagonals with the checkerboard/vertically

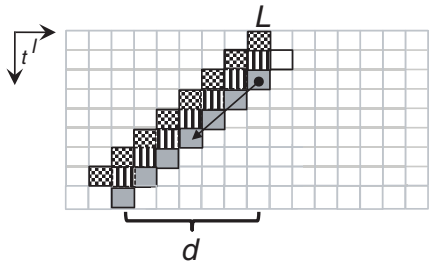

Figure 4. Diagonal values involved in HPF updating of the dark gray diagonal.

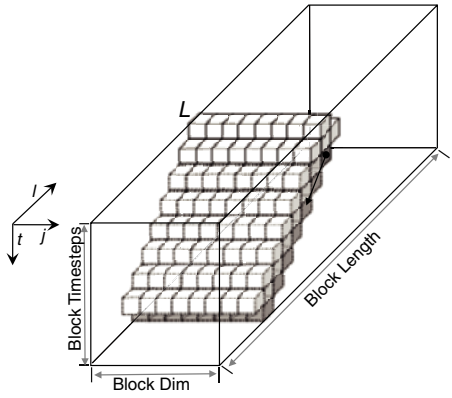

Figure 5. Diagonals stored in shared memory for a *chunk of block timesteps* and a thread block of size *Block Dim*.

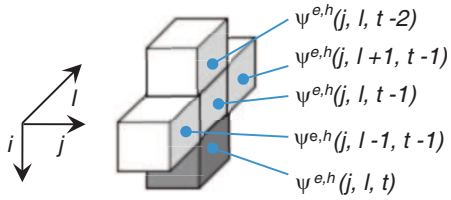

Figure 6. HPF stencil 3D representation.

striped patterns, and to the white cell black bordered. Moreover, we observed that each diagonal's update requires the points used by the previous diagonal on its left. In fact the next diagonal in $l = L + 1$ uses the values of the diagonals with dark grey and striped patterns. Therefore, keeping these two previous diagonals in shared memory, thread can avoid $2d$ accesses to the global memory. This scheme is repeated for all the next diagonals. In this way the only global memory accesses are the ones needed to update the top point in each diagonal to get the values $\psi^{e,h}(j, l + 1, t - 1)$ and $\psi^{e,h}(j, l, t - 2)$, beyond the accesses to save the results.

The shared memory is a limited resource for each SM, which is 16 KB for the Tesla architecture. Therefore the size of the diagonals to be kept in shared memory (the one being updated and the previous two) is chosen to allow the allocation of at least one block of threads per SM: Diagonals size $= Block\,Dim \times Block\,Timesteps \times 3$.

The more blocks allocate on SM the better is the scheduling performance as it will be shown in the numerical results. This new

approach allows saving global memory accesses but it requires each *thread block* to update its domain in *chunks* of limited d time steps. In Fig. 5 is shown the computational pattern for a generic diagonal updating a chunk of thickness $d = Block\,Timesteps$. The computation is thus organized in chunks of size *Block Timesteps* along the time coordinate, for each thread block, and results are stored in GPU global memory.

The last resource constraint regards the choice of GPU slice size to allow computed data to be held in GPU global memory. Moreover, in order to save space and bandwidth we encoded the permittivity mask in a bitmask for each dielectric constant, requiring $O(\log_2 n_x) \times n_y$ memory instead of $O(n_x) \times n_y$.

4. NUMERICAL RESULTS AND DISCUSSIONS

The HPF tool is used to simulate the 2D behavior of a device with nanostructures organized in different geometric configurations (Fig. 7). We show the speedup using the GPUs and how the choice of thread block dimension and the presence of permittivity variations

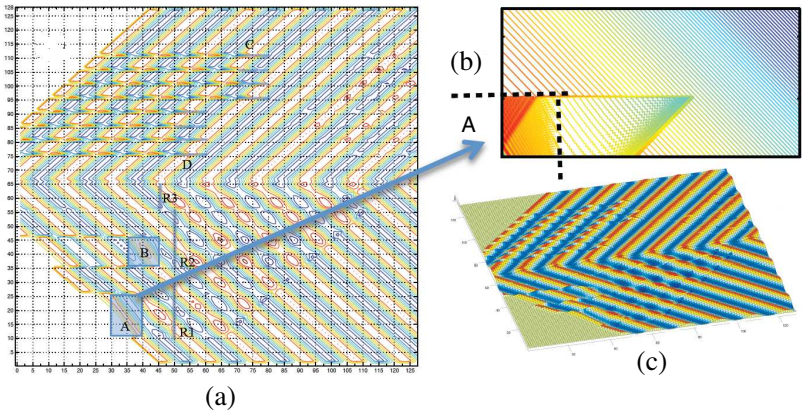

Figure 7. (a) Total electric field scattered by the nanodefects A, B, C, D, R_1, R_2, and R_3 at time step $t = 150$: Part of the electric field is back scattered and part is propagated along the waveguide. The source is sinusoidal with a wavelength of $\lambda = 1.55\,\mu$m. The rods C are $1\,\mu$m in length. (b) Zoomed image of the total electric field confined in proximity of the dielectric corner of A: The nanoscale dimension of the zoomed rectangular dielectric corner is $50\,$nm $\times$ $50\,$nm. (c) 3D view of (a).

(the background is air with refractive index $n_{air} = 1$ and the dielectric defect is a material with refractive index $n_{defect} = 1.414$) may bias the performances.

In the simulations we considered below a computational space of 1024×1024 points with a resolution of $dx = dy = 1\,\text{nm}$ is used. The time step size of $dt = dy \times \sqrt{(\varepsilon_{air}\mu_0)}$ is chosen to satisfy the stability constraints CFL [22] and the source is sinusoidal working at $\lambda = 1.55\,\mu\text{m}$.

We compared the performance of the HPF tool for GPU with the previous implementation for CPU (Fig. 8) through the Speedup metric: *Speedup=(Sequential execution time)/(Parallel execution time)*. Where the *Sequential execution time* is the execution time of the HPF version for CPU while *parallel execution time* is the execution time of the GPU version.

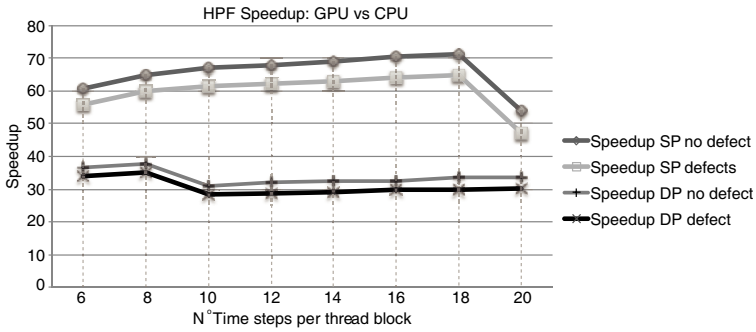

Figure 8. Speedup of the GPU (NVIDIA GeForce GTX295) vs the CPU (Intel Core2 Quad CPU Q9550) is evaluated varying the Time-step size in four different configuration: single(SP)/double(DP) precision, presence (defect)/absence (no defect) of dielectric.

The GPU computing system we used is taken from the mainstream computing market. It is a 64 bit Linux workstation equipped with an Intel Core2 Quad Q9550 CPU running at 2.83 GHz, 4 GB of DRAM memory and a NVIDIA GeForce GTX295 graphics card.

We run the serial code on the CPU without exploiting multithreading whilst the CUDA version has been run on only one of the two GPUs of the GTX295 connected to its 896MB DRAM. The version of the CUDA toolkit used to develop the application is the 3.10 while the driver installed in the operating system is that of NVIDIA v.256.40.

In the speedup graph shown in Fig. 8 we compare the performances of the same device simulation varying the size of *block timesteps* and

keeping fixed the Block dim size to 32 threads (see Fig. 5). We adopted four different configurations using single/double precision and presence/absence of permittivity discontinuities (i.e., defects). Detailed simulation time and speedup are reported in Table 1.

Table 1. Time and Speedup for the simulation on GPU and CPU of a discretized domain of 1024×1024 points for 192 timesteps. Data are shown varying the *block timesteps size*, precision and presence of defects. Underlined values are the best configurations.

	Double precision						Single precision					
	Without defect			With defect			Without defect			With defect		
Block timesteps	Time [ms] CPU	GPU	Speedup	Time [ms] CPU	GPU	Speedup	Time [ms] CPU	GPU	Speedup	Time [ms] CPU	GPU	Speedup
6	99543	2730	36.5	99553.5	2916	34.1	98210	1623	60.5	98932	1770	55.9
8	99543	2640	**37.7**	99553.5	2844	**35.0**	98210	1515	64.8	98932	1647	60.1
10	99543	3219	30.9	99553.5	3507	28.4	98210	1464	67.1	98932	1614	61.3
12	99543	3102	32.1	99553.5	3462	28.8	98210	1446	67.9	98932	1596	62.0
14	99543	3081	32.3	99553.5	3405	29.2	98210	1425	68.9	98932	1575	62.8
16	99543	3051	32.6	99553.5	3345	29.8	98210	1395	70.4	98932	1542	64.2
18	99543	2958	33.7	99553.5	3321	30.0	98210	1380	**71.2**	98932	1527	**64.8**
20	99543	2949	33.8	99553.5	3315	30.0	98210	1827	53.8	98932	2106	47.0

In the speedup graph (Fig. 8) we observe that using single precision performs better than double precision. Generally this is due to the greater number of clock cycles needed by the SMs to perform operations on double precision floating-point numbers. In our test case, as in other algorithms [21], the bandwidth is the bottleneck that does not allow to harness all the available computational power. Therefore the different performance depends on the fact that the smaller is the data size the more elements are available for computation at a fixed bandwidth (single precision floating point numbers are 4 Bytes in size whilst the doubles are 8 Bytes).

In the case of a device embedding nanostructures, a lower performance is observed in the speedup graph. In fact when a warp of threads deals with a discontinuity, it must use different permittivity constants through a branch. In CUDA different execution paths of a branch are serialized on a SM (i.e., branch divergence) yielding a loss of performance. We considered here the worst case with a configuration with maximum number of divergent warps vs a device without defects with no divergence.

The last consideration concerns the decrease of performance, shown in Fig. 8, when changing the *block timesteps* size from 18 to

20 timesteps in the single precision simulation and from 8 to 10 in the double precision. The reason is associated with the number of blocks and therefore the number of threads that can be allocated on the SMs. In fact the memory required by the allocation of a block depends on the block timesteps size, the bigger is the block timesteps the bigger is the memory required. Therefore, when block timesteps exceeds 8, the amount of blocks that can be allocated on a SM decreases from two to one in the case of double precision. In the case of single precision, if the size of time block exceeds 18, the blocks allocated on a SM decrease from two blocks to one. The amount of blocks exceeds the available SMs thus the scheduling is worse.

The results show that a good pattern of computing and the careful choice of shared memory usage can provide a speedup of 70 × in single precision and almost 40 × in double precision, Fig. 8.

5. CONCLUSIONS

The goal of this paper is, starting from a defined code, to find a GPU matching approach. In particular the HPF code is performed in order to understand the limits of the GPU for similar EM codes.

The CUDA platform and architecture has a low learning curve that allows every scientist to benefit of the extreme performance of the GPU computing. Nevertheless, smart tailoring of the code/algorithm to the resources available is required.

In this work we demonstrate that with a few code modifications it is possible to perform simulations that otherwise require an expensive HPC cluster.

ACKNOWLEDGMENT

D. T. would like to thank Dr. F. Calabi of the Istituto Nanoscienze-CNR, and Prof. G. Aloisio of the University of Salento in Lecce, for their support.

REFERENCES

1. Massaro, A., M. Grande, R. Cingolani, A. Passaseo, and M. De Vittorio, "Design and modelling of tapered waveguide for photonic crystal slab coupling by using time-domain Hertzian potential formulation," *Opt. Express*, Vol. 15, No. 25, 16484–16499, 2007.

2. VanRoey, J., J. van Derdonk, and P. Lagasse, "Beam-propagation method: Analysis and assessment," *J. Opt. Soc. Am.*, Vol. 71, 803–810, 1981.
3. Yee, K. S., "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. Antennas Propagation*, Vol. 14, No. 8, 302–307, 1966.
4. Massaro, A. and T. Rozzi, "Rigorous time-domain analysis of dielectric optical waveguides using Hertzian potentials formulation," *Opt. Express*, Vol. 14, No. 5, 2027–2036, 2006.
5. Massaro, A., V. Tasco, M. T. Todaro, R. Cingolani, M. De Vittorio, and A. Passaseo, "Scalar time domain modeling and coupling of second harmonic generation process in gaas discontinuous optical waveguides," *Opt. Express*, Vol. 16, No. 19, 14496–14510, 2008.
6. Owens, J. D., M. Houston, et al., "GPU computing," *Proceedings of the IEEE*, Vol. 96, No. 5, 879–899, May 2008.
7. Nickolls, J. and W. J. Dally, "The GPU computing era," *IEEE Micro.*, Vol. 30, 56–69, 2010.
8. Huang, S., S. Xiao, and W. Feng, "On the energy efficiency of graphics processing units for scientific computing," *IEEE International Symposium on Parallel & Distributed Processing*, 1–8, 2009.
9. Krakiwsky, S. E., L. E. Turner, and M. M. Okoniewski, "Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU)," *IEEE MTT-S Int. Microwave Symp. Digest*, 1033–1036, 2004.
10. Peng, S. X. and Z. P. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Trans. Antennas and Propagation*, Vol. 56, No. 7, 2130–2133, Jul. 2008.
11. Zainud-Deen, S. H., E. El-Deen, M. S. Ibrahim, K. H. Awadalla, and A. Z. Botros, "Electromagnetic scattering using gpu-based finite difference frequency domain method," *Progress In Electromagnetics Research B*, Vol. 16, 351–369, 2009.
12. Jiang, W. Q., M. Zhang, and Y. Wang, "CUDA-based radiative transfer method with application to the EM scattering from a twolayer canopy model," *Journal of Electromagnetic Waves and Applications*, Vol. 24, No. 17–18, 2509–2521, 2010.
13. Xu, K., Z. H. Fan, D. Z. Ding, and R. S. Chen, "GPU accelerated unconditionally stable Crank-Nicolson FDTD method for the analysis of three-dimensional microwave circuits," *Progress In Electromagnetics Research*, Vol. 102, 381–395, 2010.

14. Mur, G., "Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic field equations," *IEEE Trans. Electromagn. Compat.*, Vol. 23, 377–382, 1981.
15. Taflove, A. and S. C. Hagness, *Computational Electrodynamic: The Finite-difference Time-domain Method*, 2nd edition, Chaps. 2, 4, and 7, Arthec House Publishers, London, 2000.
16. Lindholm, E, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE Micro.*, Vol. 28, 39–55, 2008.
17. NVIDIA CUDA C Programming guide v.3.2. Nvidia Corp., 2010.
18. NVIDIA CUDA C Best Practices guide v.3.2. Nvidia Corp., 2010.
19. Patterson, D. A. and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 4th edition, Morgan Kaufmann, 2008.
20. Kirk, D. B. and W.-M. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann, 2010.
21. Ryoo, S., C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-M. W. Hwu, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM*, 73–82, New York, USA, 2008.
22. Courant, R., K. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," *IBM Journal of Research and Development*, Vol. 11, No. 2, 215–234, 1967.