Graphics Processing Unit Implementation of Multilevel Plane-Wave Time-Domain Algorithm

Yang Liu, *Student Member, IEEE*, Abdulkadir C. Yücel, Vitaliy Lomakin, *Senior Member, IEEE*, and Eric Michielssen, *Fellow, IEEE*

***Abstract*—A graphics processing unit implementation of the multilevel plane-wave time-domain algorithm for rapidly evaluating transient electromagnetic fields generated by large-scale dipole constellations is proposed. The implementation achieves 50× speedup and 60%–75% memory reduction compared to its serial CPU implementation.**

***Index Terms*—Graphics processing unit (GPU), multilevel plane-wave time-domain (ML-PWTD) algorithm, NVIDIA CUDA fast Fourier transform library (CUFFT).**

I. Introduction

THE MULTILEVEL plane-wave time-domain (ML-PWTD) algorithm is an efficient and accurate scheme for evaluating transient electromagnetic (EM) fields produced by large-scale dipole constellations. It reduces the computational complexity and memory requirements of evaluating transient EM fields from N_s dipoles for N_t time-steps from $O(N_tN_s^2)$ and $O(N_s^2)$ to $O(N_tN_s\log^2 N_s)$ and $O(N_s^{1.5})$, respectively [1], [2]. When used in tandem with classical marching-on-in-time schemes for solving time-domain integral equations, it permits the fast and accurate analysis of scattering from, and radiation by, complex and large-scale structures. Despite its favorable computational complexity and memory requirements, the computational cost of serial ML-PWTD CPU implementations limits their applicability to the analysis of real-world phenomena. To this end, substantial efforts have been directed toward parallelizing the ML-PWTD algorithm on CPU clusters [3], [4].

Recently, graphics processing units (GPUs) that perform ultra-fast floating-point operations on multithreaded many-core processors have been shown to favorably compete with CPUs for parallelizing a wide variety of computational electromagnetics (CEM) algorithms [5], [6]. Here, we present an efficient GPU implementation of the ML-PWTD algorithm (henceforth termed GPU-ML-PWTD) that parallelizes *all* stages of the ML-PWTD scheme, ranging from near-field calculation to construction and translation of outgoing rays, processing and projection of incoming rays, and vector interpolation and filtering operations; the present work extends a previous implementation that only parallelized two of these stages [7]. The proposed GPU-ML-PWTD algorithm is successfully applied to the computation of transient EM fields generated by large-scale dipole constellations.

Manuscript received June 08, 2014; revised July 18, 2014 and August 11, 2014; accepted August 14, 2014. Date of publication August 22, 2014; date of current version January 15, 2015. This work was supported in part by the AFOSR/NSSEFF Program under Award FA9550-10-1-0180 and the National Science Foundation (NSF) and the Office of Naval Research under Award N00014-11-1-0720.

Y. Liu, A. C. Yücel, and E. Michielssen are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: liuyangz@umich.edu; acyucel@umich.edu; emichiel@umich.edu).

V. Lomakin is with the Department of Electrical and Computer Engineering, University of California, San Diego, CA 92093 USA (e-mail: vlomakin@ucsd.edu).

Color versions of one or more of the figures in this letter are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LAWP.2014.2350967

II. Direct Scheme and ML-PWTD Algorithm

This section summarizes the main features of the ML-PWTD scheme and introduces notation pertinent to the description of its GPU implementation in Section III. For a detailed description of the ML-PWTD scheme, the reader is referred to [1] and [2]. Let S denote an arbitrarily shaped surface supporting the current density $\mathbf{J}(\mathbf{r},t)$. The time-differentiated electric (E) field generated by $\mathbf{J}(\mathbf{r},t)$ is

$$\partial_t \mathbf{E}(\mathbf{r},t) = (\mu_0/(4\pi))\left(\partial_t^2 \mathcal{I} - c^2\nabla\nabla\right) \cdot \int_S \mathbf{J}(\mathbf{r}',t)\,\delta(t-R/c)/R dS' \tag{1}$$

where ∂_t denotes the time derivative, μ_0 and c are the free-space permeability and speed of light, $\mathcal{I}$ is the identity dyad, $R = |\mathbf{r}-\mathbf{r}'|$ is the distance between source point $\mathbf{r}'$ and observation point $\mathbf{r}$, and $\delta(\cdot)$ is the delta Dirac function. Assume that $\mathbf{J}(\mathbf{r},t)$ is approximated by N_s surface-bound point dipoles as

$$\mathbf{J}(\mathbf{r},t) = \sum\nolimits_{n=1}^{N_s} f_n(t)\delta(\mathbf{r}-\mathbf{r}_n)\,\hat{\mathbf{u}}_n \qquad \forall \mathbf{r}\in S. \tag{2}$$

Here, $\mathbf{r}_n$ and $\hat{\mathbf{u}}_n$ are the nth dipole's position and direction, and $f_n(t)$ is its temporal signature, which is band-limited to maximum frequency $\omega_{\max} = 2\pi f_{\max}$ and quasi-time-limited to $0 < t < T$. To evaluate interactions between these dipoles, the temporal signature $f_n(t)$ oftentimes is discretized as

$$f_n(t) = \sum\nolimits_{j=1}^{N_t} I_{n,j}T_j(t) \tag{3}$$

where $T_j(t) = T(t-j\Delta t)$ is the time-shifted Lagrange interpolant [8] and $\Delta t = \pi/\chi_t\omega_{\max}$ denotes the time-step size with oversampling factor $5 < \chi_t < 20$ and $N_t = T/\Delta t$. Substituting (2) and (3) into (1) and computing fields (excluding self-interactions) at $t_i = i\Delta t$ yields

$$\bar{\mathbf{F}}_i = \sum\nolimits_{j=0}^{i} \bar{\bar{\mathbf{Z}}}_j\bar{\mathbf{I}}_{i-j} \tag{4}$$

where the entries of $\bar{\mathbf{F}}_i$, $\bar{\mathbf{I}}_i$, and $\bar{\bar{\mathbf{Z}}}_j$ are $\hat{\mathbf{u}}_m \cdot \partial_t \mathbf{E}(\mathbf{r}_m, i\Delta t)$, $I_{n,i}$, $m, n = 1, \ldots, N_s$, and

$$Z_{j,mn} = (\mu_0/(4\pi))\hat{\mathbf{u}}_m \cdot (\partial_t^2 \mathcal{I} - c^2 \nabla\nabla) \cdot \hat{\mathbf{u}}_n T(t - |\mathbf{r} - \mathbf{r}_n|/c)/|\mathbf{r} - \mathbf{r}_n||_{t=t_j, \mathbf{r}=\mathbf{r}_m}, \quad m \neq n \quad (5)$$

with $Z_{j,nn} = 0$. Direct computation of E-fields along N_s dipoles [via (4)] requires $O(N_s^2)$ memory and $O(N_t N_s^2)$ operations for N_t time-steps. These computational requirements can be reduced to $O(N_s^{1.5})$ and $O(N_t N_s \log^2 N_s)$ via the ML-PWTD algorithm.

In the ML-PWTD scheme, a fictitious box that encloses S is recursively subdivided into eight smaller boxes a total of $N^v - 1$ times [1], [2]; boxes of the same dimension are said to belong to the same level v, $v = 1, \ldots, N^v$. There exist approximately $4^{N^v - v}$ nonempty boxes (i.e., groups) at level v, each of which can be enclosed by a sphere of radius $R^v = 2^{v - N_v} R^{N_v}$ with $R^{N_v} = O(N_s^{0.5})$. Starting from level N^v (coarsest level), two groups α and α' centered about $\mathbf{r}_\alpha^c$ and $\mathbf{r}_{\alpha'}^c$ residing at the same level v form a "far-field" pair if: 1) the distance between their centers exceeds a certain threshold, $R_{c,\alpha\alpha'} = |\mathbf{R}_{c,\alpha\alpha'}| = |\mathbf{r}_\alpha^c - \mathbf{r}_{\alpha'}^c| > \gamma R^v$ $(3 \leq \gamma \leq 6)$; and 2) their parent boxes do not constitute a far-field pair. Groups at level 1 (finest level) that do not constitute a "far-field" pair form a "near-field" pair. Contributions to (4) stemming from interactions between dipoles belonging to near-field group pairs are computed directly by (5). To evaluate contributions to (4) due to interactions between dipoles belonging to a far-field group pair at level v, a local approximate prolate spheroidal (APS) function $T^{\mathrm{APS}}(t)$ (see the reference in [2]) that is band-limited to $\omega_s = \chi_t \omega_{\max}$ and approximately time-limited to $-p_f \Delta t < t < p_f \Delta t, 5 \leq p_f \leq 10$, is used to approximate current temporal signatures. Specifically, the time signature of the nth dipole in group α at level v is broken into N_l^v consecutive band-limited subsignals as

$$f_n(t) = \sum_l^{N_l^v} f_n^l(t) = \sum_l^{N_l^v} \sum_{j=(l-1)M^v+1}^{lM^v} I_{n,j} T_j^{\mathrm{APS}}(t) \quad (6)$$

where $T_j^{\mathrm{APS}}(t) = T^{\mathrm{APS}}(t - j\Delta t)$ and $N_l^v M^v = N_t$; M^v is chosen such that the duration of each subsignal, $T^v = (M^v + 2p_f)\Delta t$, is less than $(R_{c,\alpha\alpha'} - 2R^v)/c$. To compute the time-differentiated E-field along the mth dipole (in group α') generated by the nth dipole (in group α) for the lth subsignal, first a set of outgoing rays (of group α) in directions $\hat{\mathbf{k}}_{pq}^v$ is constructed by the convolution of the projection function $\mathbf{P}_n^+(\hat{\mathbf{k}}_{pq}^v, t, \hat{\mathbf{u}}_n)$ with the subsignal $f_n^l(t)$ as

$$\mathbf{G}_{l,\alpha}^+\left(\hat{\mathbf{k}}_{pq}^v, t\right) = \sum_{n \in \alpha} \mathbf{P}_n^+\left(\hat{\mathbf{k}}_{pq}^v, t, \hat{\mathbf{u}}_n\right) * f_n^l(t). \quad (7)$$

Here, the ray directions $\hat{\mathbf{k}}_{pq}^v = (\cos\phi_q^v \sin\theta_p^v, \sin\phi_q^v \sin\theta_p^v, \cos\theta_p^v$, $p = 0, \ldots, K^v$, $q = -K^v, \ldots, K^v$, fall along $(K^v + 1)$ directions along θ and $(2K^v + 1)$ directions along ϕ, where $K^v = \lfloor 2\chi_s \omega_s R^v/c \rfloor + 1$; χ_s is the spherical oversampling factor. Next, the outgoing rays (of group α) are translated into incoming rays (of group α') by the convolution of $\mathbf{G}_{l,\alpha}^+(\hat{\mathbf{k}}_{pq}^v, t)$ with the translator $\mathcal{T}(\hat{\mathbf{k}}_{pq}^v, t)$ as

$$\mathbf{G}_{l,\alpha'}^-\left(\hat{\mathbf{k}}_{pq}^v, t\right) = \mathcal{T}\left(\hat{\mathbf{k}}_{pq}^v, t\right) * \mathbf{G}_{l,\alpha}^+\left(\hat{\mathbf{k}}_{pq}^v, t\right). \quad (8)$$

Finally, incoming rays are projected onto the mth dipole by convolving the projection function $\mathbf{P}_m^-(\hat{\mathbf{k}}_{pq}^v, t, \hat{\mathbf{u}}_m)$ with the incoming rays and summing over all directions with quadrature weights ω_{pq} as

$$\hat{\mathbf{u}}_m \cdot \partial_t \mathbf{E}(\mathbf{r}_m, t) = \sum_{p=0}^{K^v} \sum_{q=-K^v}^{K^v} w_{pq} \left[\mathbf{P}_m^-\left(\hat{\mathbf{k}}_{pq}^v, t, \hat{\mathbf{u}}_m\right)\right]^T * \mathbf{G}_{l,\alpha'}^-\left(\hat{\mathbf{k}}_{pq}^v, t\right). \quad (9)$$

In (7)–(9)

$$\mathbf{P}_{\{m,n\}}^{\pm}(\hat{\mathbf{k}}_{pq}^v, t, \hat{\mathbf{v}}) = \hat{\mathbf{k}}_{pq}^v \times \hat{\mathbf{v}} \delta\left(t \pm \hat{\mathbf{k}}_{pq}^v \cdot \left(\mathbf{r}_{\{m,n\}} - \mathbf{r}_{\{\alpha',\alpha\}}^c\right)/c\right) \Big/ 4\pi \quad (10)$$

$$\mathcal{T}(\hat{\mathbf{k}}_{pq}^v, t) = \frac{\mu_0 \partial_t^3}{R_{c,\alpha\alpha'}} \sum_{k=0}^{K^v} (2k+1) \Phi_k\left(\frac{ct}{R_{c,\alpha\alpha'}}\right) \Phi_k\left(\frac{\hat{\mathbf{k}}_{pq}^v \cdot \mathbf{R}_{c,\alpha\alpha'}}{R_{c,\alpha\alpha'}}\right) \quad (11)$$

where $\Phi_k(\cdot)$ is the Legendre polynomial of degree k and $|t| \leq R_{c,\alpha\alpha'}/c$. In practice, only outgoing/incoming rays of finest level groups are constructed/projected directly from/onto dipoles using (7)/(9); those of higher level groups are computed via the global vector spherical interpolation/filtering [2].

III. GPU-ML-PWTD Algorithm

The ML-PWTD scheme for computing dipole interactions consists of four stages: 1) calculation of near-field interactions via classical methods; 2) construction of outgoing rays; 3) translation of outgoing rays into incoming rays; and 4) processing and projection of incoming rays. These stages are interleaved by: 5) global interpolation and filtering operations. A viable GPU implementation must comprehensively tackle all five computational components of the scheme, which complicates its development compared to other CEM schemes [5], [6]. We delineate key ideas that guided the implementation of these five computational components before embarking on their detailed description.

An NVIDIA GPU consists of many streaming multiprocessors (SMs), each of which contains multiple cores. Under the Compute Unified Device Architecture (CUDA) framework, these SMs can execute a multithreaded function, termed kernel. A group of 32 threads forms a basic execution unit, dubbed a warp that is dynamically scheduled to one SM. Warps are further combined into blocks such that all threads in one block perform similar tasks that can be synchronized. Moreover, the GPU provides several types of memory, including shared and global units. Shared memory, which is private to one block, has small capacity and low latency, while global memory, which is accessible by all blocks, has large capacity but higher latency. To alleviate any performance degradation due to the use of high-latency global memory, the following overarching strategies drove the development of our GPU-ML-PWTD implementation: 1) Coalescing memory access of all threads in a warp to contiguous memory addresses by careful arrangement

of threads/blocks as well as input/output data layouts. 2) Hiding latency by scheduling more warps while a single warp in the SM is accessing the global memory. This is achieved by assigning sufficiently large number of warps (or equivalently threads and blocks). 3) Minimizing the memory usage by storing only necessary quantities (and calculating all others on the fly) and parallelizing loops that require the least global memory access.

A. Near-Field Calculation

The interactions between dipoles in near-field groups are computed by (5) for every time-step via launching one GPU kernel. This operation can be performed on a GPU by parallelizing loops over source and observation groups, as well as source dipoles and observer dipoles in each group. Note that multiple writing to one entry of $\bar{\mathbf{F}}_i$ [in (4)] is required at each iteration of the loops, which are over source groups and source dipoles in one group. To this end, the loops that are over observer dipoles in a group and observation groups are parallelized via a "one thread per observer dipole" and "one block per observation group" strategy, producing coalesced global memory access. In this strategy, each thread in one block computes the interactions between the observer dipole (that the thread is responsible for) and source dipoles in all source groups in the near-field interaction list of the observation group. To this end, threads collectively load $\mathbf{r}_n$ and $\hat{\mathbf{u}}_n$ of dipoles in each near-field pair (stored in contiguous spaces in global memory) into their shared memory, calculate (5) on the fly, and update the pertinent entry of $\bar{\mathbf{F}}_i$.

B. Construction of Outgoing Rays

The outgoing rays of finest level groups are computed by (7) for every M^1 time-steps by launching one GPU kernel. This computation can be carried out on a GPU by parallelizing loops over temporal samples of a subsignal, dipoles of a group, directions, and groups. Note that multiple access to the memory occupied by one outgoing ray is required in each iteration of the loops over temporal samples of subsignals and dipoles of a group. For that reason, the loops that are over directions and groups are parallelized via a "one thread per direction" and "one block per group" strategy. In such a strategy, the threads in each block collectively load $\mathbf{r}_n$ and $\hat{\mathbf{u}}_n$ of dipoles in one group into their shared memory, calculate the APS interpolants on the fly, use them to project $I_{n,j}$ onto one ray, and sum the projections of all dipoles (in the source group) to compute (7).

C. Translation of Outgoing Rays into Incoming Rays

Translation between far-field group pair (α, α') at level v is performed by (8) for every ξM^v time-steps via launching GPU kernels, where ξ is a constant that depends on $R_{c,\alpha\alpha'}$. This operation is executed on a GPU for each pair separately by the following steps (Fig. 1).

1) The lth outgoing ray of group α, $\mathbf{G}^+_{l,\alpha}(\hat{\mathbf{k}}^v_{pq}, t)$ (depicted by a rectangular block in Fig. 1), is Fourier transformed to the frequency domain, $\hat{\mathbf{G}}^+_{l,\alpha}(\hat{\mathbf{k}}^v_{pq}, \omega)$.
2) The Fourier transform of the translator $\hat{\mathcal{T}}(\hat{\mathbf{k}}^v_{pq}, \omega)$ is directly computed in the frequency domain on approximately ξM^v samples (see [2] for analytical expressions) and then multiplied with the Fourier transform of the outgoing rays of group α, $\hat{\mathbf{G}}^+_{l,\alpha}(\hat{\mathbf{k}}^v_{pq}, \omega)$; both operations are parallelized via a "one thread per frequency sample" and "one block per direction" strategy, thereby producing coalesced memory access.
3) The resulting data is inverse Fourier transformed into the time domain and the l incoming ray of group α', $\mathbf{G}^-_{l,\alpha'}(\hat{\mathbf{k}}^v_{pq}, t)$, is updated.

In steps 1 and 3, Fourier transforms are performed by the batched CUDA fast Fourier transform (CUFFT) library that allows simultaneous execution of $(K^v+1)(2K^v+1)$ FFTs; the transforms are accelerated by extending the sizes of frequency and temporal sequences to powers of two by zero padding.

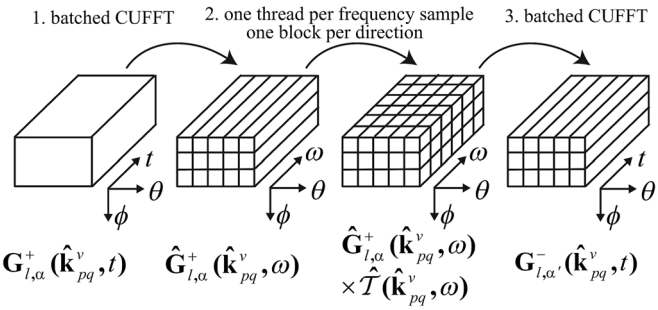

Fig. 1. GPU implementation of one translation operation.

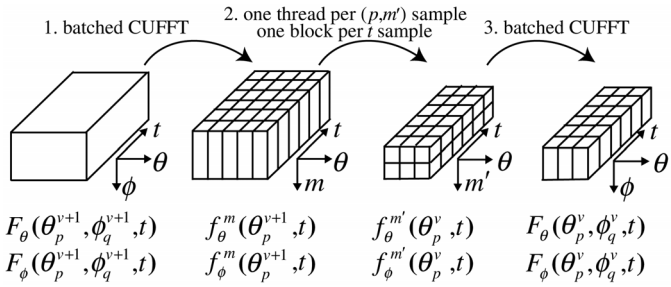

Fig. 2. GPU implementation of one spherical filtering operation.

D. Processing and Projection of Incoming Rays

The incoming rays of finest level groups are projected onto the dipoles by (9) for every time-step by launching a GPU kernel. This projection can be performed on a GPU by parallelizing the loops that are over directions, dipoles of a group, and groups. Since multiple access to the memory space of a dipole of a group is required at each iteration of the loop over directions, loops that are over dipoles of a group and loops over groups are parallelized by a "one thread per dipole" and "one block per group" strategy. Each thread calculates the APS interpolants for all directions and uses them to update fields along the dipole.

E. Spherical Interpolation/Filtering

The outgoing/incoming rays of groups at higher levels (i.e., $v > 1$) are computed using the global vector spherical interpolation/filtering scheme of [2]. (Note: Here, only the GPU implementation of spherical filtering to obtain incoming rays is explained for the sake of brevity as that interpolates outgoing rays is very similar.) The incoming rays of a

group at level v consist of two transverse components, i.e., $\mathbf{G}^{-}_{l,\alpha'}(\hat{\mathbf{k}}^{v}_{pq},t) = F_\theta(\theta^v_p,\phi^v_q,t)\hat{\boldsymbol{\theta}} + F_\phi(\theta^v_p,\phi^v_q,t)\hat{\boldsymbol{\varphi}}$. These components are obtained by filtering the transverse components of the incoming rays of the parent group at level $v+1$, which are $F_\theta(\theta^{v+1}_p,\phi^{v+1}_q,t)$ and $F_\phi(\theta^{v+1}_p,\phi^{v+1}_q,t)$. This filtering operation is performed on a GPU by the following steps (Fig. 2).

1) The forward FFTs of $F_\theta(\theta^{v+1}_p,\phi^{v+1}_q,t)$ and $F_\phi(\theta^{v+1}_p,\phi^{v+1}_q,t)$ are computed along the ϕ-dimension.
2) The Fourier coefficients, $f^m_\theta(\theta^{v+1}_p,t)$ and $f^m_\phi(\theta^{v+1}_p,t)$, $m=1,\ldots,2K^{v+1}+1$, are truncated in the spectral domain via fast spectral truncation and correction [2] and the truncated Fourier coefficients, $f^{m'}_\theta(\theta^v_p,t)$ and $f^{m'}_\phi(\theta^v_p,t)$, $m'=1,\ldots,2K^v+1$, are obtained. The truncation and correction operations are parallelized via a "one thread per (p,m') sample" and "one block per t sample" strategy; each thread calculates $f^{m'}_\theta(\theta^v_p,t)$ and $f^{m'}_\phi(\theta^v_p,t)$ for one θ^v_p and one m'; again, this procedure yields coalesced memory access.
3) $F_\theta(\theta^v_p,\phi^v_q,t)$ and $F_\phi(\theta^v_p,\phi^v_q,t)$ are obtained by inverse FFTing $f^{m'}_\theta(\theta^v_p,t)$ and $f^{m'}_\phi(\theta^v_p,t)$.

Note that GPU kernels performing these steps are launched for each group separately. In steps 1 and 3, FFTs are performed by the batched CUFFT library, which allows simultaneous execution of $M^{v+1}(K^{v+1}+1)$ forward FFTs and $M^{v+1}(K^v+1)$ inverse FFTs.

IV. Numerical Results

This section presents several numerical tests that demonstrate the accuracy and efficiency of the proposed GPU-ML-PWTD algorithm. All tests involve a set of dipoles that are randomly oriented and located on square plates with edge length ranging from 1.5 m ($N_s = 2500$) to 9 m ($N_s = 40\,000$). The dipoles' temporal signature is $f_n(t) = m_n\cos(2\pi f_0(t-6\sigma))\exp(-(t-6\sigma)^2/2\sigma^2)$, where $\sigma = 4/(2\pi f_{\max} - f_0)$, $f_0 = 800$ MHz, $f_{\max} = 1$ GHz, and m_n is a random real number between 0 and 1. The magnitudes of the time derivatives of the E-fields along the dipoles, $|F_{i,m}|$, $m=1,\ldots,N_s$, $i=1,\ldots,N_t$, are computed for $N_t = 500$ time-steps with $\Delta t = 6.25\times10^{-2}$ ns. In what follows, the GPU and CPU implementations of the direct scheme are termed GPU-Direct and CPU-Direct, respectively. GPU and serial CPU implementations (double precision) are executed on a Tesla C2050 device and an Intel Xeon E5-2670, respectively.

First, for an arbitrarily selected dipole, the $|F_{i,m}|$ obtained by CPU-Direct, CPU-ML-PWTD, and GPU-ML-PWTD schemes are compared ($N_s = 10\,000$) (Fig. 3). The L^2 norm error of the E-field computed by the GPU-ML-PWTD scheme is 1×10^{-4} (compared to the exact value obtained by the CPU-Direct scheme), while the relative difference between the E-field values obtained by the CPU-ML-PWTD and GPU-ML-PWTD schemes is around machine precision.

Second, the computational time for each stage of CPU-ML-PWTD and GPU-ML-PWTD schemes is tabulated for $N_s = 2500$ and $N_s = 40\,000$ (Table I). Note that the computational time for GPU-ML-PWTD scheme includes the time of data transfer that is performed at the beginning and end of each stage. As N_s increases, the speedup achieved by GPU-ML-PWTD scheme at each stage increases due to larger number of threads and blocks that are being leveraged.

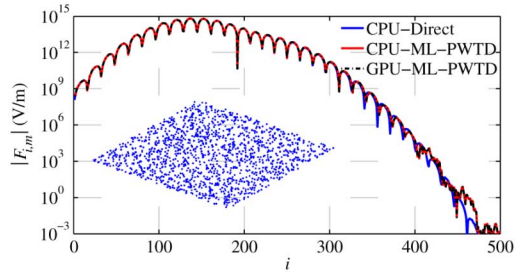

Fig. 3. Comparison of $|F_{i,m}|$ obtained by CPU-direct, CPU-ML-PWTD, and GPU-ML-PWTD schemes.

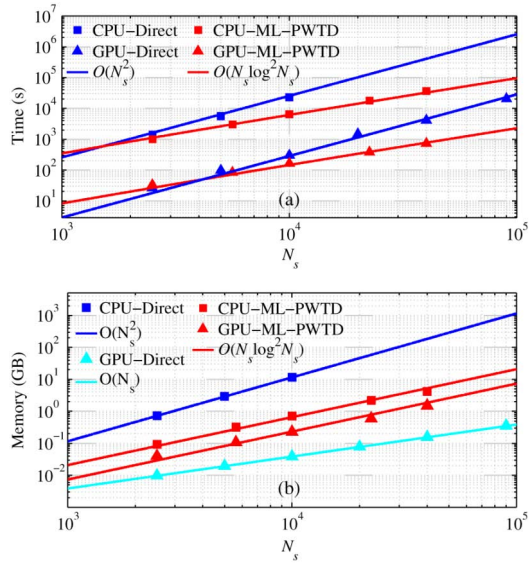

Fig. 4. (a) Total computation time and (b) memory requirement of CPU-Direct, GPU-Direct, CPU-ML-PWTD, and GPU-ML-PWTD schemes.

TABLE I
Computation Time for Each Stage of CPU-ML-PWTD and GPU-ML-PWTD Schemes (in Seconds) and the Ratio Between Them

	$N_s = 2500$			$N_s = 40{,}000$		
	CPU	GPU	Ratio	CPU	GPU	Ratio
Outgoing ray const.	11.5	2.1	5.5	201.6	30.8	6.7
Interpolation	2.3	0.2	11.5	332.1	23.1	14.3
Translation	194.6	6.8	28.6	17,883.1	380.4	47.1
Filtering	1.2	0.1	12.0	313.5	22.2	14.1
Incoming ray proj.	13.3	3.6	3.7	153.8	38.5	4.0
Near-field calc.	783.4	20.2	38.8	20,275.9	239.7	84.8
Total	998.8	32.9	30.4	39,247.1	736.8	53.3

Finally, the overall computational time and memory required by the CPU-Direct, GPU-Direct, CPU-ML-PWTD, and GPU-ML-PWTD schemes are compared for increasing N_s (Fig. 4). Here, the parallelization strategy described in Section III-A was applied to the GPU-Direct scheme. The GPU-Direct scheme achieves 54.4×–76.3× speedup, while GPU-ML-PWTD scheme achieves 30.4×–53.3× speedup, and outperforms the other three schemes as N_s increases [Fig. 4(a)]. As the GPU-Direct scheme computes matrix elements of $\bar{\bar{\mathbf{Z}}}_j$ on the fly as opposed to the CPU-Direct scheme that precalculates

them, it requires $O(N_s)$ global memory [Fig. 4(b)]. (Note: Calculating $\bar{\bar{\mathbf{Z}}}_j$ on the fly would result in dramatically higher computation time for the CPU-Direct scheme.) On the other hand, GPU-ML-PWTD scheme achieves substantial memory reduction compared to CPU-ML-PWTD scheme since it only stores ray data at one or two levels. The maximum number of sources for the GPU-ML-PWTD scheme is $N_s = 40\,000$ and limited by the 3-GB global memory. In addition, the performance of all schemes complies with the theoretical complexities.

V. Conclusion

An implementation of ML-PWTD algorithm that executes all PWTD stages on a GPU was presented. The proposed implementation achieves 50× speedup and up to 75% memory reduction compared to its CPU counterpart. Efforts to embed the proposed implementation within classical marching-on-in-time-based integral equation solvers and hybridization with CPU-parallel methods are underway.

References

[1] A. A. Ergin, B. Shanker, and E. Michielssen, "The plane-wave time-domain algorithm for the fast analysis of transient wave phenomena," *IEEE Antennas Propag. Mag.*, vol. 41, no. 4, pp. 39–52, Aug. 1999.

[2] B. Shanker, A. A. Ergin, M. Lu, and E. Michielssen, "Fast analysis of transient electromagnetic scattering phenomena using the multilevel plane wave time domain algorithm," *IEEE Trans. Antennas Propag.*, vol. 51, no. 3, pp. 628–641, Mar. 2003.

[3] N. Liu, M. Lu, B. Shanker, and E. Michielssen, "The parallel plane wave time domain algorithm-accelerated marching on in time solvers for large-scale electromagnetic scattering problems," in *Proc. IEEE Int. Symp. AP-S/URSI*, 2004, pp. 4212–4215.

[4] Y. Liu, H. Bağcı, and E. Michielssen, "Solving very large scattering problems using a parallel PWTD-enhanced surface integral equation solver," in *Proc. IEEE Int. Symp. AP-S/URSI*, 2013, p. 106.

[5] S. Li, R. Chang, A. Boag, and V. Lomakin, "Fast electromagnetic integral-equation solvers on graphics processing units," *IEEE Antennas Propag. Mag.*, vol. 54, no. 5, pp. 71–87, May 2012.

[6] J. Guan, S. Yan, and J. M. Jin, "An OpenMP-CUDA implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-GPU computing systems," *IEEE Trans. Antennas Propag.*, vol. 61, no. 7, pp. 3607–3616, Jul. 2013.

[7] Y. Liu, V. Lomakin, and E. Michielssen, "Graphics processing unit-accelerated implementation of the plane wave time domain algorithm," in *Proc. 28th Annu. Rev. Prog. Appl. Comput. Electromagn.*, 2012, pp. 1–6.

[8] G. Manara, A. Monorchio, and R. Reggiannini, "A space-time discretization criterion for a stable time-marching solution of the electric field integral equation," *IEEE Trans. Antennas Propag.*, vol. 45, no. 3, pp. 527–532, Mar. 1997.