CMP-722 Please cite as:
D. S. Abraham and D. D. Giannacopoulos, "A Parallel Finite-Element Time-Domain Method for Nonlinear Dispersive Media," in IEEE Transactions on Magnetics, vol. 56, no. 2, pp. 1-4, Feb. 2020, Art no. 7507604, doi: 10.1109/TMAG.2019.2952528.

A Parallel Finite-Element Time-Domain Method for Nonlinear Dispersive Media

David S. Abraham[1], *Student Member, IEEE*, Dennis D. Giannacopoulos[1], *Senior Member, IEEE*
[1]Department of Electrical & Computer Engineering, McGill University, Montréal, Québec H3A 0E9 Canada

In this paper, a novel use of Graphics Processing Units (GPUs) is presented for the acceleration of Finite-Element Time-Domain (FETD) methods containing electrically complex media. By leveraging the massively parallel architecture of the GPU via NVIDIA's Compute Unified Device Architecture (CUDA) language, the immense computational burden imposed by these materials can be largely alleviated, facilitating their modeling and incorporation into electromagnetic devices and systems. To that end, an analysis of both mixed and vector wave equation-based nonlinear dispersive FETD algorithms is presented in order to both identify computational bottlenecks and determine their amenability to parallelization. Based on this analysis, a parallel elemental matrix evaluation procedure is proposed, which when coupled to the recently derived Gaussian Belief Propagation Method for matrix assembly and solution, demonstrates up to a 200 times performance increase as compared to a traditionally serial implementation.

Index Terms—**Dispersion, Finite-Element Analysis, Nonlinear Optics, Parallel Processing.**

I. Introduction

With an ever increasing number of connected devices in today's world, the ubiquity and importance of telecommunications infrastructure cannot be overstated. Indeed, now more than ever, the need for fast, reliable, and efficient means of communication are spurring research and development of electromagnetic devices for the transmission and reception of data. In particular, developments in the field of nonlinear optics, propelled by advances in fabrication techniques and material sciences, have resulted in large-scale improvements in speed, capacity, and reliability. With an increasing reliance of these optical devices on complex electromagnetic material interactions, such as dispersion and nonlinearity, it is unsurprising that demand for design tools capable of accurately modeling these effects has also increased, as cost-effective alternatives to expensive empirical study.

To that end, two numerical techniques based upon the Finite-Element Time-Domain (FETD) method have recently been proposed for the treatment of electrically complex media [1], [2]. These full-wave methods have proven capable of solving the dispersive and nonlinear Maxwell's Equations without simplification or assumptions about the underlying wave solutions. The first method does so within the context of a mixed FETD framework, using both edge and face elements, whereas the second adopts the more familiar vector wave equation (VWE) approach, using only edge elements. Both methods have to date demonstrated remarkable success, and are capable of modeling very general permittivity functions in an accurate, stable, and versatile way.

Despite the successes of these methods, however, their main drawback undoubtedly lies in the immense computational burden they pose. Thus, despite their many desirable properties these simulations often lay outside the realm of feasibility for even modestly sized problems. Under these circumstances, any attempt to improve the speed and efficiency of these dispersive nonlinear FETD methods, and thus alleviate the principle burden they pose, could pave the way for fast and accurate solutions to problems containing electrically complex media. This is the principle goal of this paper.

While many approaches are possible to decrease the execution time of these algorithms, the focus of this paper is on the use of parallelization via Graphics Processing Units (GPUs). In the following sections, these algorithms will be analyzed to both qualify and quantify performance bottlenecks, as well as determine their amenability to parallelization. These findings are then used to devise a parallel implementation on an NVIDIA GPU, leveraging several novel and existing techniques. The algorithm is then benchmarked against a serial implementation and demonstrates substantial speedups.

II. FETD for Electrically Complex Media

For many electrically complex materials an adequate permittivity model which incorporates linear dispersion, instantaneous nonlinearity, and dispersive nonlinearity, is given by the following constitutive equation:

$$\vec{D} = \epsilon_0 \Big[1 + \chi^{(1)}(t) * \vec{E} + \chi^{(3)} \left(\alpha E^2 + (1-\alpha) g(t) * E^2 \right) \vec{E} \Big] \quad (1)$$

where $\chi^{(1)}$ models linear dispersion, $\chi^{(3)}$ an instantaneous Kerr and/or dispersive stimulated Raman nonlinearity, $E = |\vec{E}|$, and $*$ denotes convolution [3].

As described in [1] and [2], the constitutive equation in (1) can be substituted into Maxwell's Equations, and a standard variational approach applied to obtain a semi-discrete system. In the case of the method in [1], the equations are treated directly as a coupled first-order system and discretized using both edge and face elements. In contrast, the method in [2]

Manuscript received ~~December 1, 2012~~; revised ~~August 26, 2015~~; accepted ~~July 16, 2019~~. Corresponding author: D. S. Abraham (email: david.abraham@mail.mcgill.ca).

operates on the second-order vector wave equation, requiring the use of edge elements only.

While the methods in question do have several notable differences, it has been noted on multiple occasions that the two formulations produce remarkably similar final update equations and procedures [2], [4]. Due to this striking resemblance, any analysis carried out on one of these algorithms is likely to be immediately applicable to other, with very minor, if any, modification. As a result, for the remainder of this paper focus will be placed on the mixed formulation, though again the VWE equations may easily be substituted in the ensuing discussion.

Following through with the mixed formulation, including a temporal discretization via Crank-Nicolson, results in the following update equation for the electric field:

$$\left([K]^{n+1} + \frac{\Delta t^2}{4}[C]^T[M_f][C]\right)\{e\}^{n+1} = \left([K]^n - \frac{\Delta t^2}{4}[C]^T[M_f][C]\right)\{e\}^n + \Delta[C]^T[M_f]\{b\}^n - \left(\{\mathcal{W}_1\}^n - \{\mathcal{W}_1\}^{n-1}\right) + \frac{\Delta t}{2}(\{g\}^{n+1} + \{g\}^n) \quad (2)$$

in which $\{e\}$ are the electric field weights, $\{b\}$ the magnetic field weights, Δt the discrete time step size, $\{g\}$ a volume current source term, $\{\mathcal{W}_1\}$ an auxiliary variable used in the modeling of linear dispersion (see [4]), $[C]$ a connectivity matrix composed of ones and zeros, and the remaining elemental matrices given by:

$$[K]^n_{ij} = \int_\Omega \big(a_0 + \alpha\epsilon_0\chi^{(3)}(E^2)^n + (1-\alpha)\epsilon_0\chi^{(3)}\cdot [h_0(E^2)^n + \mathcal{G}_1^n]\big)\vec{W}_i^{(1)}\cdot\vec{W}_j^{(1)}d\Omega \quad (3)$$

$$[M_f]_{ij} = \int_\Omega \frac{1}{\mu}\vec{W}_i^{(2)}\cdot\vec{W}_j^{(2)}d\Omega \quad (4)$$

where a_0 is associated with the linear dispersion, h_0 and $\mathcal{G}_1$ with the nonlinear dispersion, μ is the permeability, and $\vec{W}^{(n)}$ represents a vector n-form basis function.

Since the $[K]$ matrix depends on the field strength E via the permittivity, equation (2) is inherently nonlinear. As a result, [1] and [2] implement a nonlinear Newton-Raphson iteration in order to update the electric field at each time step:

$$\{e\}^{n+1}_{(k+1)} = \{e\}^{n+1}_{(k)} - [J]^{-1}_{(k)}\{f\}_{(k)} \quad (5)$$

where $\{f\}$ represents equation (2) with all terms collected on the left-hand side, and which also requires the derivation and implementation of the following Jacobian matrix at each iteration:

$$[J]^n_{ij} = [K]_{ij} + \frac{\Delta t^2}{4}[C]^T[M_f][C]_{ij} + \int_\Omega 2\epsilon_0\chi^{(3)}(\vec{W}_i^{(1)}\cdot\vec{E}^n)(\vec{W}_j^{(1)}\cdot\vec{E}^n)d\Omega. \quad (6)$$

When combined with suitable update equations for the convolution variables and magnetic field, equations (2), (5), and (6) provide the necessary framework for updating the electric field within complex media.

III. Algorithm Analysis

The main overhead associated with the above algorithms naturally comes from the introduction of both dispersion and nonlinearity into the formulations. As originally detailed in [4], and implemented in [1] and [2], dispersion is principally handled via the z-transform, which introduces a series of auxiliary variables which must be updated at each time step. However, the overhead imposed by these auxiliary variables, as well as possible parallelization strategies, has already been discussed in [5]. As a result, here the focus will primarily be on the nonlinearity, which likely dwarfs the dispersion in terms of computational effort.

As mentioned earlier, nonlinear FETD methods tend to be significantly more computationally intensive than their linear counterparts. An initial overview of equations (2), (5), and (6) easily shows that for each time step, the nonlinear algorithm will require multiple solutions of the Jacobian matrix equation, as compared to the single matrix solution required in linear computations. Moreover, there is an additional complicating factor: the dependence of the $[K]$ and $[J]$ matrices on the unknown solution $\{e\}^{n+1}$ causes equations (2), (5), and (6) to change not only at each time step, but at each new iteration of the Newton-Raphson algorithm within each time step. Thus, every nonlinear elemental $[K]$ and $[J]$ matrix must be recomputed and reassembled into their global counterparts multiple times per time step, prior to each new matrix solve.

Further complicating matters is that, unlike in the linear case, in general no closed-form expressions are available for the elemental $[K]$ and $[J]$ matrices. As a result, each one must be numerically evaluated via Gaussian quadrature, significantly exacerbating the computational burden.

IV. Parallelization Strategy

Having potentially identified the main computational bottlenecks in the previous section, here a number of strategies are developed for their parallelization. While efficient implementations of nonlinear algorithms have previously been derived which may address some of the above concerns, such as the Jacobian-Free Newton-Krylov method [6], here a different approach is adopted. More specifically, rather than avoiding the overhead associated with the Newton-Raphson iteration via approximation, here the full non-approximative implementation is undertaken and the overhead directly parallelized.

The first issue of elemental matrix evaluation is relatively straightforward to address. Within the framework of the Finite-Element method, the local elemental matrices are fundamentally independent from one another, with elements only "sharing" information once these are assembled into their global counterparts. As a result, the evaluation of these elemental matrices constitutes an embarassingly parallel problem, where multiple workers can each compute an elemental matrix with zero need to communicate. This type of problem is ideal for implementation on a GPU, and is visually depicted in Figure 1, where each GPU worker or thread has been assigned an element. While relatively simple to implement, the impact of parallelizing the matrix evaluation in this way cannot be overstated. Indeed, due to the large computational burden

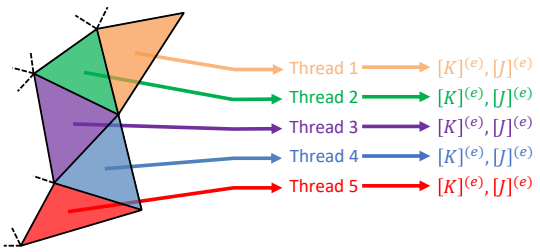

Fig. 1. Parallelization of elemental matrix evaluation.

imposed by millions of applications of Gaussian quadrature, significant speedup is expected to come from this step alone.

The matters of global matrix assembly and solution, however, are not so simple. The parallel assembly of global matrices from their local counterparts has been widely studied within Finite-Elements [7], [8]. As mentioned earlier, it is during assembly that elements share information, which can lead to race conditions and undefined behaviour. To mitigate this, it is often necessary for these methods to adopt additional layers of processing, such as coloring algorithms. Moreover, parallel algorithms for the solution of matrix equations constitutes a massive area of study in its own right, with techniques such as parallel preconditioned conjugate gradient achieving notable success [9]. Rather than address these two issues individually, here a different approach will be adopted wherein *both* will be addressed simultaneously.

Recently, the Gaussian Belief Propagation (GaBP) algorithm has been proposed for the treatment of time-dependent problems within the FETD method [10], [11]. The GaBP algorithm is remarkable in this application not only because it solves the matrix equations in parallel, but because it does so *without* ever assembling the global matrices. The ability of the GaBP algorithm to work only with local matrices and solve the equivalent global problem at the same time makes it an ideal candidate for the dispersive nonlinear algorithms above.

In essence, the GaBP algorithm works by converting the global matrix equation into a maximization problem of a multivariate Gaussian probability density function, which equates to finding the mean. Moreover, due to the Gaussian nature of the distribution, it is possible to factor it in such a way as to maintain the mean value or solution. Thus, rather than having one global distribution to maximize there are a series of local factors, each associated with an individual element, whose overall product should be as large as possible.

Naturally the shared nature of unknowns between elements will require some communication between them to jointly maximize their likelihoods. The Finite-Element GaBP algorithm then proceeds roughly as follows:

1) Each element receives messages α_i from their neighbors, containing information about the likely values of its unknowns
2) Each element uses these messages, as well as information from its own local matrices, to update these values and produce its own estimates
3) Each element then sends messages β_i containing updated estimates to each of its neighbors.

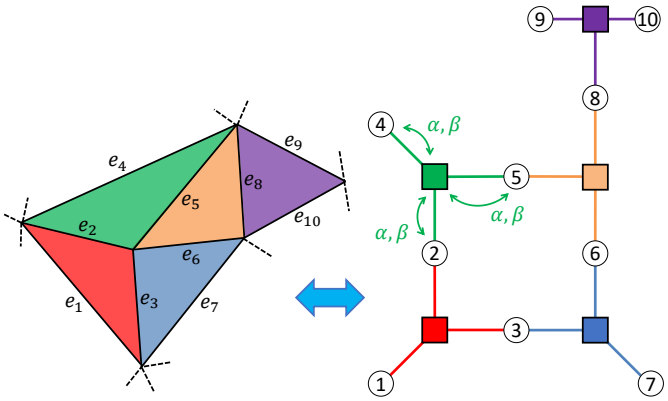

Fig. 2. Schematic representation of the Finite-Element GaBP algorithm.

4) The process continues iteratively until convergence.

The algorithm is schematically depicted in Figure 2 above, with more information available in [10] and [11]. By implementing this GaBP technique for the nonlinear dispersive FETD algorithms above, along with the parallelization described in Figure 1, each of the major predicted bottlenecks can hopefully be largely alleviated.

V. Implementation

Due to the nature of GPU architecture, a haphazard implementation of a particular algorithm can easily result in unwanted serialization of operations. Thus the performance of any particular algorithm heavily depends on its implementation. To that end, the algorithms above were implemented on an NVIDIA GPU using the Compute Unified Device Architecture (CUDA) language, with particular emphasis on efficiency and parallelism. Specifically, attention was given to two criteria: coalesced global memory access and minimization of memory transfers between the host and GPU [12]. Coalesced memory accesses were obtained by re-ordering the problem's data structures to coincide with GPU access patterns. Lastly, while matrix evaluation, assembly, and solution make up the majority of the problem, in order to avoid costly memory transfers between the host and GPU, every other intervening operation was also implemented on the GPU as well. As a result, after sending initial mesh and problem data to the GPU, the remainder of the algorithm could proceed with minimal host intervention. One drawback to this, however, is that the entire computation must fit into GPU memory.

VI. Results

To test the effectiveness of the above parallel algorithms, a test problem was devised for which the novel GPU implementation could be benchmarked against a standard serial algorithm. The chosen problem is a classic demonstration of a nonlinear phenomena known as a spatial soliton, in which the diffraction of a beam propagating in a bulk medium is negated by nonlinearity. The selected domain was a two-dimensional rectangle measuring 30 cm wide by 1 m long with perfect electric conductor boundaries, implemented with the mixed FETD formulation. The domain was uniformly filled with

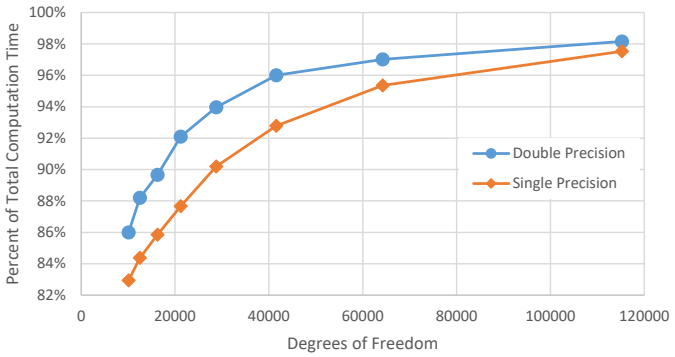

Fig. 3. Matrix evaluation and assembly as a function of degrees of freedom.

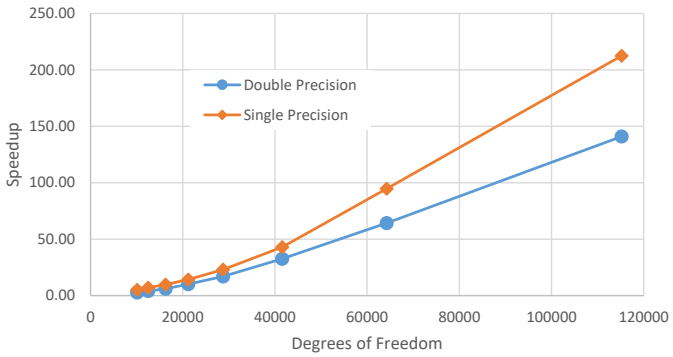

Fig. 4. GPU speedup as a function of degrees of freedom.

a bulk instantaneous nonlinear medium ($\alpha = 0$) for which $\chi^{(1)} = 2.2$ and $\chi^{(3)} = 1.5 \times 10^{-19}\ m^2/V^2$. While the choice of a two dimensional problem was mostly for convenience, it is worth noting that three dimensional performance should be equal to or even exceed the results shown here, due to better GPU utilization. However, as mentioned, the increased number of variables in the 3D case must still fit within the global memory of the GPU being used. The serial algorithm was compiled in C++ and executed on a single core of an Intel 8700K CPU, clocked at 3.7 GHz. The CPU code was optimized via the \O2 compiler flag, as well as by enabling intrinsic functions. The GPU algorithm, in contrast, was compiled in CUDA v9.2 and executed on an NVIDIA GTX 1070Ti with 2432 cores clocked at 1607 MHz and 8 GB of GDDR5 memory. This is roughly sufficient memory for the product of the number of degrees of freedom and time steps to equal just less than 10^9.

To test the hypothesized bottlenecks in Section III, the serial algorithm was profiled to verify which subroutines were most costly. These results are shown in Figure 3, in which the proportion of computation time devoted to elemental matrix calculation and global matrix assembly are plotted as a function of degrees of freedom, for a series of refined meshes. It is clear that together these postulated bottlenecks do dominate the simulation, more so than matrix solving.

Figure 4 shows the result of using the aforementioned parallel algorithms to alleviate these bottlenecks. As the problem size grows, better utilization of GPU resources is achieved, and performance steadily increases. In the case of double precision, the highest speedup observed was about 141 times faster than serial execution, whereas in the case of single precision, it was an impressive 213 times faster. This represents a reduction in computation time from roughly 1.5 hours down to 30 seconds.

VII. Conclusion

In conclusion, the suspected bottlenecks associated with computationally intensive nonlinear dispersive Finite-Element Time-Domain methods have been identified and characterized. Moreover, it has been shown that by using a new mixture of a parallel elemental matrix evaluation method as well as a novel application of GaBP to nonlinear FETD problems, that these computational burdens can be largely alleviated, via the use of Graphics Processing Units. With peak execution 213 times faster than a traditional implementation, these results represent a substantial improvement with potential consequences not only for the study of electrically complex media, but any nonlinear numerical method with a similar formulation, such as those for magnetic machines. Lastly, given the success of the GPU algorithm, future work could also include investigation of other implementations, such as parallel CPU algorithms.

Acknowledgment

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for their support.

References

[1] D. S. Abraham and D. D. Giannacopoulos, "A convolution-free mixed finite-element time-domain method for general nonlinear dispersive media," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 1, pp. 324 – 334, Jan. 2019.

[2] D. S. Abraham and D. D. Giannacopoulos, "A convolution-free finite-element time-domain method for the nonlinear dispersive vector wave equation," *IEEE Transactions on Magnetics*, accepted for publication.

[3] R. M. Joseph and A. Taflove, "FDTD maxwell's equations models for nonlinear electrodynamics and optics," *IEEE Transactions on Antennas and Propagation*, vol. 45, no. 3, pp. 364–374, Mar. 1997.

[4] A. Akbarzadeh-Sharbaf and D. D. Giannacopoulos, "A stable and efficient direct time integration of the vector wave equation in the finite-element time-domain method for dispersive media," *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 1, pp. 314–321, Jan. 2015.

[5] D. S. Abraham and D. D. Giannacopoulos, "Dispersive möbius transform finite-element time-domain method on graphics processing units," *IEEE Transactions on Magnetics*, vol. 52, no. 3, Mar. 2016.

[6] D. A. Knoll and D. E. Keyes, "Jacobian-free newton-krylov mehotds: A survey of approaches and applications," *Journal of Computational Physics*, vol. 193, no. 2, pp. 357–397, 2004.

[7] U. Kiran, D. Sharma, and S. S. Gautam, "Gpu-warp based finite element matrices generation and assembly using coloring method," *Journal of Computational Design and Engineering*, Nov. 2018, in press, DOI: 10.1016/j.jcde.2018.11.001.

[8] Y. V. Khalevitsky, N. V. Burmasheva, and A. V. Konovalov, "An approach to the parallel assembly of the stiffness matrix in elastoplastic problems," in *AIP Conference Proceedings*, 2016.

[9] R. Helfenstein and J. Koko, "Parallel preconditioned conjugate gradient algorithm on gpu," *Journal of Computational and Applied Mathematics*, vol. 236, no. 15, pp. 3584 – 3590, Sept. 2012.

[10] D. Fernandez, A. Akbarzadeh-Sherbaf, and D. D. Giannacopoulos, "Solving finite-element time-domain problems with gabp," *IEEE Transactions on Magnetics*, vol. 53, no. 6, Jun. 2017.

[11] Y. El-Kurdi, W. J. Gross, and D. D. Giannacopoulos, "Parallel multigrid acceleration for the finite-element gaussian belief propagation algorithm," *IEEE Transactions on Magnetics*, vol. 50, no. 2, Feb. 2014.

[12] *CUDA C Programming Guide*, NVIDIA Corporation, Aug. 2019.