A Multi-GPU Accelerated DGTD Method for Solving Electrically Large-Scale Problems

Ziang Shen[1] and Lei Zhao[2]

[1]College of Electronic and Information Engineering
Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
shenziang@nuaa.edu.cn

[2]School of Information and Control Engineering
China University of Mining and Technology, Xuzhou 221116, China
leizhao@cumt.edu.cn

Abstract – In this paper, we proposed a multiple graphics processing units (GPU) platform accelerated discontinuous Galerkin time-domain (DGTD) method for solving electrically large-scale problems. Rather than simply porting the code to a GPU, we proposed a cache optimization strategy tailored to the GPU architecture. Furthermore, by grouping and reordering the elements and employing asynchronous techniques, we achieve a linear speedup ratio when scaling across multiple GPUs. The numerical examples not only validate accuracy of the proposed method, but also demonstrate excellent performance, achieving up to 40 times speedup even compared to parallelism CPU implementations.

Index Terms – Discontinuous Galerkin time-domain (DGTD) method, multiple graphics processing units (multi-GPU).

I. INTRODUCTION

Electrically large-scale problems have always been a challenge in computational electromagnetics (CEM). Their considerable computational complexity demands a combination of high-performance hardware and efficient algorithms. Among various electromagnetic numerical methods, the discontinuous Galerkin time-domain (DGTD) method [1–4] has attracted growing attention from researchers. As an efficient algorithm, DGTD offers greater flexibility than traditional approaches when dealing with complex problems, as it supports non-conformal, unstructured, and mixed-type meshes while maintaining reliable accuracy [5–7]. The numerical flux introduced by the finite-volume time-domain (FVTD) method [8] allows the boundaries between adjacent elements to be non-conformal. Moreover, models can be decomposed into several subdomains with different time iteration step increments [9, 10], facilitating the solution of multiscale problems. DGTD also exhibits strong potential for parallel computing [11–13]. During the time iteration process, computations within each element are entirely independent, making DGTD highly suitable for high-performance computing (HPC).

However, traditional CPU-based parallelization is still unable to handle such large computational loads. Consequently, GPU-based acceleration has attracted significant attention in DGTD research. In [14], DGTD was deployed on the GPU, achieving approximately 50 times speedup over serial computation. The authors of [15] implemented a hybrid MPI/GPU DGTD algorithm with local time stepping (LTS). In [16], the GPU-accelerated DGTD method was used to solve the scattering problem of electrically large targets. In [17], GPU-DGTD is used to solve hybrid meshes. The authors of [18] implemented GPU acceleration of a low-storage Runge-Kutta (LSRK) time iteration method. The authors of [19] used GPU-accelerated DGTD to simulate EM systems with field-circuit interactions. A common limitation in the above studies is that they do not fully leverage systems equipped with two or more GPU, and the use of multiple GPUs has been widely applied in other numerical methods. The authors of [20] assembled the finite-element method (FEM) matrices on both single GPU and multiple GPUs. The authors of [21, 22] discuss strategies for load balancing and for reducing cross-GPU communication. The authors of [23] solved FEM using a distributed message passing interface (MPI) approach. Unlike these algorithms, the DGTD has lower memory requirements when solving electrically large-scale problems, but its challenge lies in the massive amount of computation. Therefore, using multiple clusters actually increases unnecessary communication overhead. DGTD is better suited to a shared-memory programming model, using OpenMP to control multiple GPU devices, and also offers the practical advantage of minimal intrusion into the existing codebase. For example, a multi-GPU leap-frog scheme was proposed in [24].

Submitted On: September 03, 2025
Accepted On: November 13, 2025

This paper proposes a multi-GPU DGTD method. Combined with an asynchronous multi-GPU strategy to hide data transmission time and cache optimization strategy to minimize data access latency, our method delivers superior performance, enabling the solution of electrically large-scale problems that are intractable with traditional CPU-based approaches due to excessive computational demands.

II. FORMULATIONS AND GPU ACCELERATED IMPLEMENTATION

A. DGTD method

In the source-free region Ω, consider Maxwell's equations in the time domain:

$$\mu\frac{\partial \mathbf{H}}{\partial t} = -\nabla\times\mathbf{E}, \tag{1}$$

$$\varepsilon\frac{\partial \mathbf{E}}{\partial t} = \nabla\times\mathbf{H} - \sigma\mathbf{E}, \tag{2}$$

where ε is the permittivity, μ is the permeability, σ is the conductivity, $\mathbf{E}$ is the electric field intensity, and $\mathbf{H}$ is the magnetic field intensity. To solve equations (1) and (2), the computational domain Ω is divided into K non-overlapping tetrahedral elements as $\Omega \approx \bigcup_{k=1}^{K}\Omega_k$.

Here we express (1) and (2) as the conservation formulation [25]:

$$\mathbf{Q}\partial_t\mathbf{q} + \nabla\cdot\mathbf{F} = 0, \tag{3}$$

in which

$$\nabla\cdot\mathbf{F} = \partial_x\begin{bmatrix}\mathbf{e}_x\times\mathbf{E}\\ -\mathbf{e}_x\times\mathbf{H}\end{bmatrix} + \partial_y\begin{bmatrix}\mathbf{e}_y\times\mathbf{E}\\ -\mathbf{e}_y\times\mathbf{H}\end{bmatrix} + \partial_z\begin{bmatrix}\mathbf{e}_z\times\mathbf{E}\\ -\mathbf{e}_z\times\mathbf{H}\end{bmatrix}, \tag{4}$$

$$\mathbf{Q} = \begin{bmatrix}\mu & 0\\ 0 & \varepsilon\end{bmatrix}, \tag{5}$$

$$\mathbf{q} = \begin{bmatrix}\mathbf{H}\\ \mathbf{E}\end{bmatrix}, \tag{6}$$

where $\mathbf{e}_x$, $\mathbf{e}_y$, and $\mathbf{e}_z$ are the unitary vectors along the axis in the cartesian coordinate system.

In each tetrahedral element Ω_k, we assume that the approximation solution of (3) is $\tilde{\mathbf{q}}^k(\mathbf{r},t)$ and can be expressed as:

$$\tilde{\mathbf{q}}^k(\mathbf{r},t) = \sum_{i=1}^{N_p} q(\mathbf{r}_i,t)l_i(\mathbf{r}), \tag{7}$$

where $l_i(\mathbf{r})$ is the Lagrange polynomial basis function, N_p is the number of degrees of freedoms (DOFs) and its relationship with the number of Lagrange order n is:

$$N_p = \frac{(n+1)(n+2)(n+3)}{6}. \tag{8}$$

Since $\tilde{\mathbf{q}}^k(\mathbf{r},t)$ and $\mathbf{q}^k(\mathbf{r},t)$ are not identical, we use Galerkin's method to force the projection between the test function and residual to be zero. We have the semi-discrete system of Maxwell's equations:

$$\varepsilon\mathbf{M}^k\frac{\partial \mathbf{E}}{\partial t} = \mathbf{S}^k\times\mathbf{H} + \oint_{\partial\Omega_k}\mathbf{n}\times(\mathbf{F}_E^k - \mathbf{F}_E^*)l_i^k(\mathbf{r})d\mathbf{r}, \tag{9}$$

$$\mu\mathbf{M}^k\frac{\partial \mathbf{H}}{\partial t} = -\mathbf{S}^k\times\mathbf{E} + \oint_{\partial\Omega_k}\mathbf{n}\times(\mathbf{F}_H^k - \mathbf{F}_H^*)l_i^k(\mathbf{r})d\mathbf{r}, \tag{10}$$

where $\mathbf{M}^k$ and $\mathbf{S}^k$ are the local mass matrix and stiffness matrix, respectively, and n denotes the unit normal vector points from the element Ω_k to the neighbor element.

In the DGTD method, the fields inside each element must remain continuous, while discontinuous are allowed across the boundaries of adjacent elements. These adjacent elements are connected through the numerical fluxes, so the solving process inside each element is independent and parallelizable. There are a variety of types of numerical fluxes to choose from, and here we list only the formulation of the central numerical flux:

$$\mathbf{n}\times\mathbf{E}^* = \frac{1}{2}\mathbf{n}\times(\mathbf{E}^+ + \mathbf{E}^-), \tag{11}$$

$$\mathbf{n}\times\mathbf{H}^* = \frac{1}{2}\mathbf{n}\times(\mathbf{H}^+ + \mathbf{H}^-), \tag{12}$$

where $\mathbf{E}^-$ or $\mathbf{H}^-$ denote the field inside the element Ω_k on the boundary, and $\mathbf{E}^+$ or $\mathbf{H}^+$ denotes the field in the neighbor element on the boundary.

The equations above can be solved by various methods. Considering that the memory resources of the GPU are limited, we employ the LSRK [18] method to solve the DGTD equations:

$$\mathbf{P}_0 = \mathbf{q}_n(t)$$
$$\begin{cases}\mathbf{K}_i = a_i\mathbf{K}_{i-1} + \Delta t f(\mathbf{P}_{i-1}, t + c_i\Delta t)\\ \mathbf{P}_i = \mathbf{P}_{i-1} + b_i\mathbf{K}_i\end{cases},$$
$$\mathbf{q}_n(t+\Delta t) = \mathbf{P}_5 \tag{13}$$

where subscript i is taken to be 1, 2, 3, 4, and 5, which means the number of stages, coefficients a_i, b_i, and c_i define the properties of the LSRK, and Δt is the time step of DGTD.

B. Multi-GPU accelerated DGTD implementation

Among the various GPU programming frameworks, compute unified device architecture (CUDA) was selected as the primary platform for our multi-GPU DGTD implementation, owing to its mature architecture, comprehensive toolchain, and widespread community support. To fully leverage the floating-point computing power brought by GPUs, DGTD must be parallelized as fine-grained as possible. According to the previous section, during each iterative step, the electric and magnetic field at each Lagrange-Gauss-Lobatto (LGL) point can be computed independently. These field components, such as $\mathbf{E}_x$ and $\mathbf{E}_y$, are defined as the smallest parallel tasks, as shown in Fig. 1.

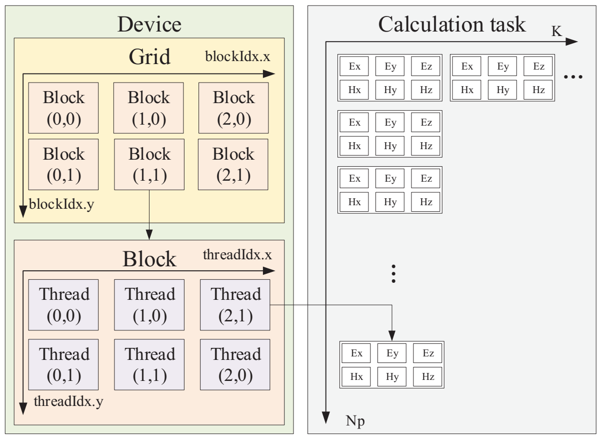


Fig. 1. Computing tasks of six scalar fields on one LGL point are packed as the minimum parallel unit and distributed to a Thread of a CUDA kernel function.

When using multiple GPU devices, it is better to use GPUs with the same specifications to ensure balanced computational load distribution. Maintaining a balanced load across devices is important to avoid a situation where one device is working while others are idle. Especially when the complex problems require a large number of iterations, the efficiency loss cannot be ignored. Assume that the computing platform has N_{device} CUDA devices. During the iterative process, the host will launch N_{device} CPU Threads, and each GPU is assigned to its corresponding CPU Thread. Each GPU is assigned an equal share of the total tasks, which is $(K \times N_p)/N_{device}$.

The key distinction between multi-GPU and single-GPU execution lies in the data dependency for numerical flux calculations. Each element requires field data from its neighbors to compute the numerical flux. When using a single GPU to execute time step iterations, the data can be stored in the device memory, and there is no need to exchange data with the host memory during the iteration. However, when using multiple GPUs for iterations, if the neighboring elements are in the same device, they can be directly obtained, but if they are in different devices, they need to be obtained from the storage of other devices. Consequently, in each time step of iteration, although the calculation time of multi-GPU is reduced, the additional data exchanges will cause performance loss.

C. Rearrange elements and asynchronous iteration

Load distribution does not simply divide the mesh elements into several equal parts and assign them to different devices. Because the mesh generated by the pre-processing software is not always ordered, direct distribution will cause a large amount of information to be exchanged between devices. Therefore, we need to reduce the number of exchanged elements as much as possible when grouping mesh elements. In this paper, the mesh is partitioned into several equal sections along its longest axis, as shown in Fig. 2.

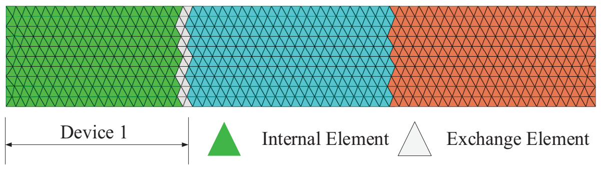


Fig. 2. Schematic diagram of the elements grouping of the target. Elements that are not connected to other groups are called internal elements, otherwise they are called exchange elements. Elements in the same group will be assigned to the same CUDA device.

During each iteration, the data of the exchange elements in each group needs to be transferred to other devices. While grouping only reduces the number of these elements, the storage order of exchange and internal elements becomes interleaved. Since data transfer between GPU and host is most efficient when performed as a single contiguous Block, we rearrange the element order within each group. In this paper, all data of exchange elements within a group are placed at the head of the array, as shown in Fig. 3.

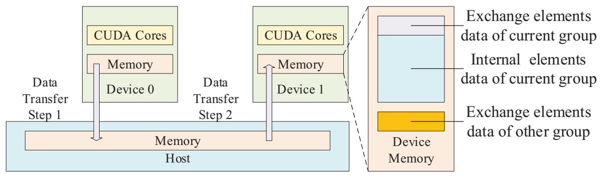

CUDA Cores
Memory
Device 0
CUDA Cores
Memory
Device 1
Data Transfer Step 1
Data Transfer Step 2
Memory
Host
Device Memory
Exchange elements data of current group
Internal elements data of current group
Exchange elements data of other group

Fig. 3. Schematic diagram of data transfer during time step iterations.

Since the data transfer is significantly slower than computation, reducing only the amount of exchanged data does not yield satisfactory acceleration. However, the asynchronous characteristics of CUDA provide a solution by allowing the GPU to overlap computation with data transmission. To leverage this, we propose an asynchronous iteration strategy. In each time step, the exchange elements are computed first, and then different CUDA streams are used to simultaneously execute two tasks: transferring the exchange elements data and computing the internal elements. Because the number of internal elements is much larger than the exchange elements, the time of data exchange can be covered by the calculation time, thereby masking the latency of data exchange.

D. Cache optimization strategy

The low memory consumption of DGTD mitigates the constraints of limited GPU memory capacity, thereby leveraging its advantages in speed and bandwidth. Furthermore, this section introduces an on-chip caching strategy designed to further accelerate data access.

As shown in Fig. 1, when a CUDA kernel function is launched, the kernel initializes a Grid containing multiple Blocks. These Blocks are distributed to streaming multiprocessors (SMs) for execution. Within each Block, Threads can transmit data through on-chip Shared Memory, which offers lower latency than global memory. In our implementation, all tasks on one element are packed as a Block, which means they can share the same set of geometric and material data, and only need to access the global memory once. In addition, we utilize texture memory to accelerate data retrieval from global memory. By binding the texture references to the corresponding global memory regions, CUDA cores can access the required data in fewer clock cycles. Additionally, the data that has been accessed will be stored in the on-chip cache, avoiding accessing the global memory again, which further optimizes memory performance.

III. NUMERICAL RESULTS

In this section, we will use several numerical examples to demonstrate the accuracy and efficiency of the multi-GPU DGTD method proposed in this paper.

A. Cache optimization and asynchronous data exchange strategy

The first example will be used to verify the accuracy and efficiency of the multi-GPU DGTD in solving radiation problems. As shown in the Fig. 4, we built a 50*60 mm patch antenna model and divided into 24080 tetrahedron elements. The substrate material has $\varepsilon = 4.4$ and $\mu = 1$. We solved the S11 parameters from 1 GHz to 10 GHz. As can be seen from the Fig. 4, the proposed method also has reliable accuracy compared with software HFSS [26] in solving radiation problems.

In this example, the performance of our strategy was evaluated on a Dell workstation equipped with three Nvidia Quadro K6000 GPUs. We first use one GPU to test our cache optimization strategy, and the computation times are shown in Table 1. In the baseline code without cache optimization, CUDA cores directly read and write to the device memory. In the cache-optimized version, the CUDA core will read data through the texture memory, and the data during the calculation process is temporarily stored in registers and Shared Memory and are written to the device memory after the calculation is completed, which minimizes interaction with the device memory. It can be seen from Table 1 that using cache instead of device memory as much as possible can effectively improve computing efficiency.

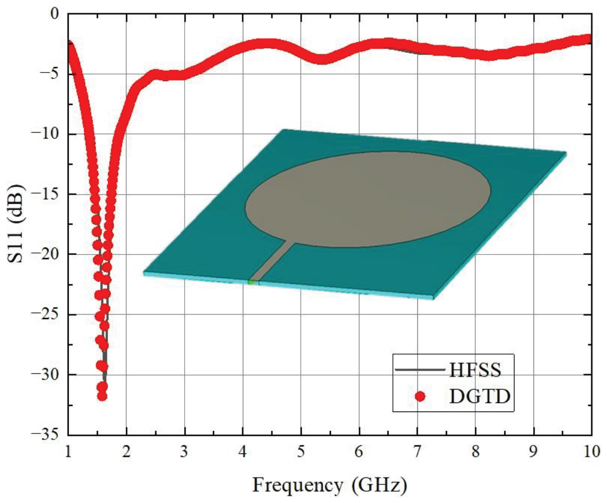


Fig. 4. Patch antenna model and comparison of numerical results of Multi-GPU DGTD and HFSS.

Table 1: Time comparison of antenna model with and without cache optimization

Method	Number of GPUs	Calc. Time (s)
no cache opt.	1	4370.1
cache opt.	1	1869.1

Table 2: Time comparison between synchronous and asynchronous data exchange

Method	Number of GPUs	Calc. Time (s)	Acc. Ratio
Synchronous	3	715.06	2.61
Asynchronous	3	687.73	2.72

Next, we validated the multi-GPU implementation based on the asynchronous data exchange strategy. The cache-optimized single-GPU program from the previous test will be used as the baseline because it does not require data exchange. The computation times of synchronous and asynchronous strategy program are compared in Table 2. Acc. Ratio in Table 2 denotes the acceleration ratio and represents the ratio of the solving speed of different methods to the baseline, calculated as the execution time of the baseline divided by our proposed approach. From the time comparison, it can be seen that multi-GPU implementation has a significant acceleration compared to single GPU. Furthermore, the efficiency improvement brought by the asynchronous strategy makes the speedup of multi-GPU implementation setups closer to linear.

B. Efficiency Comparison Between GPU and CPU

In this example, we expand the problem size to demonstrate the advantages of the multi-GPU strategy over traditional CPUs. We designed a perfect electric conductor (PEC) ship model as shown in Fig. 5. The ship model has a length of 50 m and 125477 tetrahedron elements. The bistatic radar cross-section (RCS) of the ship at 100 MHz frequency is solved and compared with the results of Feko [27] and CST [28] in Fig. 5. It can be seen that our method has reliable accuracy in calculating complex electrically large-scale targets.

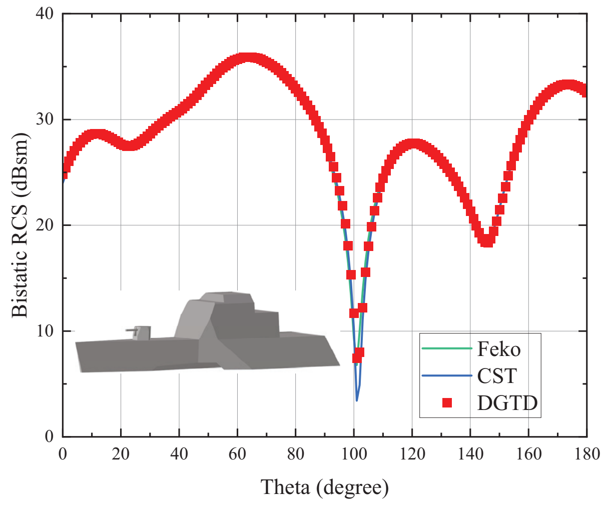


Fig. 5. Ship model and comparison of numerical results of Multi-GPU DGTD, Feko, and CST.

Table 3: Comparison of calculation time of ship model between CPU and GPU

Hardware	Calc. Time
CPU	18 h
GPU	1649.7 s

We believe that it is unfair to use products with different positioning or release times when comparing the efficiency between CPU and GPU. Therefore, in this example, the GPU program was executed by two Nvidia RTX 2080Ti, the CPU program was parallel executed by Intel i9-9900k with all eight physical cores. Hardware used in this example are flagship products of the same period. Furthermore, the GPU program is compiled by nvcc, and CPU program is compiled by icpx, which can provide more aggressive optimizations for Intel CPU to maximize performance, making them several times faster than CPU programs compiled by nvcc. From the time comparison in Table 3, we can see that the efficiency improvement of our multi-GPU DGTD is very impressive.

IV. CONCLUSION

In this paper, we proposed a multi-GPU accelerated DGTD method for solving electrically large-scale problems. Through cache optimization strategy and asynchronous data exchange strategy, we improved the efficiency of GPU programs and achieved linear speedup when using multiple devices. The several numerical examples not only prove that the proposed method is effective and accurate but also show that, when encountering complex electric large-scale problems that the CPU program cannot solve in a limited time, using GPU is the only option.

REFERENCES

[1] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods, Algorithms, Analysis, and Applications*. New York: Springer, 2008.

[2] L. E. Tobón, Q. Ren, and Q. H. Liu, “A new efficient 3D discontinuous Galerkin time-domain (DGTD) method for large and multiscale electromagnetic simulations,” *Journal of Computational Physics*, vol. 283, pp. 374–387, 2015.

[3] J. Chen and Q. H. Liu, “Discontinuous Galerkin time-domain methods for multiscale electromagnetic simulations: A review,” *Proceedings of the IEEE*, vol. 101, pp. 242–254, 2013.

[4] J. J. Jin, “From FETD to DGTD for computational electromagnetics,” *ACES Tutorial*, Williamsburg, VA, Mar. 2015.

[5] R. Sevilla, O. Hassan, and K. Morgan, “The use of hybrid meshes to improve the efficiency of a discontinuous Galerkin method for the solution of Maxwell’s equations,” *Computers & Structures*, vol. 137, pp. 2–13, 2014.

[6] S. Yan, C. P. Lin, R. R. Arslanbekov, V. I. Kolobov, and J.-M. Jin, “A discontinuous Galerkin time-domain method with dynamically adaptive cartesian mesh for computational electromagnetics,” *IEEE Transactions on Antennas and Propagation*, vol. 65, pp. 3122–3133, 2017.

[7] I. Hussain, H. Li, and Q. Cao, “Multiscale structure simulation using adaptive mesh in DGTD method,” *IEEE Journal on Multiscale and Multiphysics*

Computational Techniques, vol. 2, pp. 115–123, 2017.

[8] P. Bonnet, X. Ferrières, F. Issac, F. Paladian, J. Grando, J. Alliot, and J. Fontaine, "Numerical modeling of scattering problems using a time domain finite volume method," *Journal of Electromagnetic Waves and Applications*, vol. 11, pp. 1165–1189, 1997.

[9] J. Diaz and M. J. Grote, "Energy conserving explicit local time stepping for second-order wave equations," *SIAM Journal on Scientific Computing*, vol. 31, pp. 1985–2014, 2009.

[10] A. Taube, M. Dumbser, C. D. Munz, and R. Schneider, "A high-order discontinuous Galerkin method with time-accurate local time stepping for the Maxwell equations," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 22, pp. 77–103, 2009.

[11] L. Zhao, G. Chen, W. Yu, and J. M. Jin, "A fast waveguide port parameter extraction technique for the DGTD method," *IEEE Antennas and Wireless Propagation Letters*, vol. 16, pp. 2659–2662, 2017.

[12] G. Chen, L. Zhao, W. Yu, and J. M. Jin, "Discontinuous Galerkin time domain method for devices with lumped elements," in *International Applied Computational Electromagnetics Society* (*ACES*) *Symposium*, Suzhou, China, pp. 1–2, 2017.

[13] W. Yu, L. Zhao, and G. Chen, "A novel DGTD method and engineering applications," in *International Conference on Electromagnetics in Advanced Applications* (*ICEAA*), Cairns, QLD, Australia, pp. 324–327, 2016.

[14] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven, "Nodal discontinuous Galerkin methods on graphics processors," *Journal of Computational Physics*, vol. 228, pp. 7863–7882, 2009.

[15] S. Dosopoulos, J. D. Gardiner, and J.-F. Lee, "An MPI/GPU parallelization of an interior penalty discontinuous Galerkin time domain method for Maxwell's equations," *Radio Science*, vol. 46, 2011.

[16] L. Zhao, G. Chen, and W. H. Yu, "GPU accelerated discontinuous Galerkin time domain algorithm for electromagnetic problems of electrically large objects," *Progress in Electromagnetics Research B*, vol. 67, pp. 137–151, 2016.

[17] J. Chan, Z. Wang, A. Modave, J. F. Remacle, and T. Warburton, "GPU-accelerated discontinuous Galerkin methods on hybrid meshes," *Journal of Computational Physics*, vol. 318, pp. 142–168, 2016.

[18] Z. Shen, Z. Li, J. Yu, C. Gu, X. Chen, and L. Zhao, "GPU accelerated DGTD method for analyzing electromagnetic scattering problems," in *International Applied Computational Electromagnetics Society Symposium* (*ACES-China*), Xuzhou, China, pp. 1–3, 2022.

[19] S. Wang, Q. Zhu, Y. Wu, R. Xu, and L. Zhao, "Accelerating DGTD-based field-circuit coupling simulations using CUDA," in *Cross Strait Radio Science and Wireless Technology Conference* (*CSRSWTC*), Macao, China, pp. 1–3, 2024.

[20] H. T. Meng, B. L. Nie, S. Wong, C. Macon, and J. M. Jin, "GPU accelerated finite-element computation for electromagnetic analysis," *IEEE Antennas and Propagation Magazine*, vol. 56, no. 2, pp. 39–62, Apr. 2014.

[21] C. Richter, S. Schöps, and M. Clemens, "Multi-GPU acceleration of algebraic multi-grid preconditioners for elliptic field problems," *IEEE Transactions on Magnetics*, vol. 51, no. 3, pp. 1–4, Mar. 2015.

[22] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Communication and load balancing optimization for finite element electromagnetic simulations using multi-GPU workstation," *IEEE Transactions on Microwave Theory and Techniques*, vol. 65, no. 8, pp. 2661–2671, Aug. 2017.

[23] D. Herrero-Pérez and H. Martínez-Barberá, "Multi-GPU acceleration for finite element analysis in structural mechanics," *Appl. Sci.*, vol. 15, no. 8, 2025.

[24] T. Cabel, J. Charles, and S. Lanteri. "Multi-GPU acceleration of a DGTD method for modeling human exposure to electromagnetic waves," *Hal Inria*, p. 27, 2011.

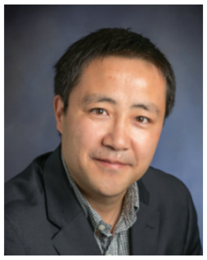
[25] B. Cockburn, S. Hou, and C.W. Shu, "The Runge–Kutta local projection discontinuous Galerkin finite element method for conservation laws. IV: The multidimensional case," *Mathematics of Computation*, vol. 54, pp. 545–581, 1990.

[26] Ansys-HFSS [Online]. Available: https://www.ansys.com/products/electronics/ansys-hfss.

[27] Altair-Feko [Online]. Available: https://www.altair.com/feko.

[28] CST-Studio-Suite [Online]. Available: https://www.3ds.com/products/simulia/cst-studio-suite.

Ziang Shen received his bachelor's degree in 2017 and his master's degree in 2020. He is presently pursuing a Ph.D. degree. His research focuses on high-performance computational electromagnetics, specifically frequency-domain, time-domain, and high-frequency asymptotic algorithms.



Lei Zhao is a professor at China University of Mining and Technology, ACES Fellow, IEEE senior member, member of the Antenna Branch of the Chinese Institute of Electronics, and chairman of IEEE AP-S Chapter Xuzhou. His main research directions are RF microwave devices, new electromagnetic materials, vortex wave communications, and computational electromagnetics.