Solving Electromagnetic Scattering Problems with Tens of Billions of Unknowns Using GPU Accelerated Massively Parallel MLFMA

Wei-Jia He, Zeng Yang, Xiao-Wei Huang, Wu Wang, Ming-Lin Yang*, and Xin-Qing Sheng, Senior Member, IEEE

Abstract—In this paper, a massively parallel approach of the multilevel fast multipole algorithm (PMLFMA) on graphics processing unit (GPU) heterogeneous platform, noted as GPU-PMLFMA, is presented for solving extremely large electromagnetic scattering problems involving tens of billions of unknowns. In this approach, the flexible and efficient ternary partitioning scheme is employed at first to partition the MLFMA octree among message passing interface (MPI) processes. Then the computationally intensive parts of the PMLFMA on each MPI process, matrix filling, aggregation and disaggregation, etc., are accelerated by using the GPU. Different parallelization strategies in coincidence with the ternary parallel MLFMA approach are designed for GPU to ensures a high computational throughput. Special memory usage strategy is designed to improve the computational efficiency and benefit data re-using. The CPU/GPU asynchronous computing pattern is designed with the OpenMP and CUDA respectively for accelerating the CPU and GPU execution parts and computation time overlapped. GPU architecture-based optimization strategies are implemented to further improve the computational efficiency. Numerical results demonstrate that the proposed GPU-PMLFMA can achieve over 3 times speed-up, compared with the 8-threaded conventional PMLFMA. Solutions of scattering by electrically large and complicated objects with about 24000 wavelengths and over 41.8 billion unknowns, are presented.

Index Terms—Multilevel fast multipole algorithm, MPI parallelization, OpenMP, CUDA, extremely large-scale problems, scattering problems

I. INTRODUCTION

In order to solve large scale electromagnetics scattering problems using the Method of Moments (MoM), various acceleration algorithms have been proposed, the Fast Multipole Method (FMM) [1] and its multi-level version the Multi-level Fast Multipole Algorithm (MLFMA) [2-3], the fast Fourier transform (FFT)-based method [4], etc. Among these algorithms, the MLFMA has been widely used due to its $O(N \log N)$ complexity, where $N$ is the number of spatial degrees of freedom. In the past few decades, many researches have been devoted to the massively parallelization of MLFMA on CPU clusters using the OpenMP, or Message-Passing Interface (MPI), or the hybrid MPI-OpenMP parallel implementations via various strategies [5-27], the simple [5], the hybrid [7-19], the hierarchical [20-24], the ternary [25], etc., subsequently increased the size of solvable problems from one million to over ten billion [25].

The CPU-based parallel implementation has advantages of large memory resources and easy to program. However, the peak computational capability of CPU, which is usually quantified by the floating-point operations per second (FLOPs) and limited by the manufacturing process, is more and more difficult to increase as the Moore’s law. Due to the special structure, the graphics processing unit (GPU), which is basically a many-core accelerator, has a much higher performance than general CPU and received more and more attention in the computational electromagnetics (CEM) community. However, since the hardware architecture of GPU is specially designed for graphics applications, special care must be taken when it is used for accelerating an algorithm as complicated as the MLFMA. Due to this reason, far fewer work on GPU acceleration of MLFMA is reported, compared with the CPU realization. In the past, progresses on how to accelerate MLFMA with GPU have been made and a series of relevant problems have been solved, GPU implementation of low-frequency MLFMA [28], Multi-GPU implementation of MLFMA using OpenMP on a single node [29], and using MPI on a GPU clusters [30-31], etc. However, the problem size reported is no larger than millions of unknowns with no more than tens of GPUs. When the problem size increases, other serious bottlenecks are encountered in the massively parallel MLFMA using GPU acceleration. How to design an efficient parallelization approach of MLFMA to make over tens of thousands of GPUs run efficiently is a challenging task.

In this paper, we present a GPU-accelerated massively parallel MLFMA, noted as GPU-PMLFMA for short, which is developed by hybridizing the MPI, the OpenMP and the compute unified device architecture (CUDA) programming model. In the GPU-PMLFMA, the MLFMA tree levels were divided into plane wave partitioning (PWP), hierarchical
structure partitioning (HSP) levels and box partitioning (BP) levels according to the ternary partitioning scheme, which has been proved able to achieve a parallel efficiency that is as high as that of the hierarchical parallelization approach, while the number of MPI processes is more flexible and not strictly limited to be $2^n$ or $4^n$ [25]. Then the computational-intensive parts on each MPI process of the ternary PMLFMA, the near and far field matrix filling in the setup stage, the aggregation and disaggregation stages in far interaction, etc., are accelerated with the aid of a GPU accelerator. In coincidence with the ternary MLFMA tree partitioning strategy, different parallelization strategies are adopted for GPU at different MLFMA tree levels. Aiming at fully using the memory and computing power of both CPU and GPU, the execution order of different stages of the PMLFMA is carefully designed. A pre-allocated chunk of memory is used on the GPU device memory to store most of the far field matrices of MLFMA, which can optimize the data transformation between GPU and CPU during iterative solution procedure, thereby significantly improve the computational efficiency and benefit data re-using. Different from former work using only GPU to accelerate the calculation, the feature of asynchronous execution between CPU and GPU is used to further reduce the actual execution time, with the OpenMP and CUDA programming models used to accelerate the CPU and GPU execution parts, respectively. Furthermore, effective GPU optimization strategies in coincidence with the special architecture, including data re-using by shared memory and parallel reduction tree-based approach, are implemented. This algorithm is shown to have a very high efficiency when solving large electromagnetic scattering problems without trading off the accuracy. To evaluate the accuracy, performance and flexibility of the proposed GPU-PMLFMA, we present various simulation results, including an aircraft model of 17254λ with 21.81 billion unknowns, and a ship model of 23940λ with 41.84 billion unknowns.

The rest of the paper is organized as follows. An overview of the ternary parallelization approach of MLFMA is provided in Section II. Section III describes the hybrid MPI, OpenMP and CUDA parallel programming model and the GPU architecture. In Section IV, the implementation of the GPU-PMLFMA is discussed in detail. Section V presents the results, followed by our concluding remarks in Section VI.

II. TERNARY PARALLELIZATION APPROACH OF THE MULTILEVEL FAST MULTIPOLAR ALGORITHM

Consider the scattering by a 3D object of arbitrary shape, when it is irradiated by an incident field, the electric field integral equation (EFIE) and the magnetic field integral equation (MFIE) can be combined together to obtain the combined field integral equation (CFIE), which exhibits both good accuracy and efficiency. The CFIE can be discretized into a dense matrix equation by using the MoM and then solved iteratively with the aid of MLFMA. In MLFMA, approximate multiple expansions of the fields are used, dividing interactions into near and far group interactions, thereby the impedance matrix $Z$ is split into two parts, $Z_{\text{near}}$ and $Z_{\text{far}}$. The interaction of near field is calculated as in the conventional MoM, while the interaction of far field is calculated in group manner performed level by level, which containing three stages, the aggregation, the translation, and the disaggregation [2]. In MLFMA, the aggregation stage is performed from the lowest level to the second highest level of the MLFMA tree. At the lowest level, the given initial value that are provided by the iterative solver are multiplied by the aggregation matrix $V_r$. Then, by interpolation and central shifting all radiation fields of child boxes at the lower level, the radiation plane waves of boxes at the higher level are obtained. This procedure continues until it reaches the second highest level of the MLFMA tree. Disaggregation stage is opposite the traversal direction of aggregation. In the disaggregation stage, both the disaggregation operations of the plane waves from parent boxes at the higher level and the translation operations of the plane waves at the same level are required. This procedure is done recursively until it reaches the lowest level. Then, the far field interaction in the current iteration can be obtained by multiplying the receiving plane waves at each smallest box with the corresponding disaggregation matrix $V_f$. All these interactions are performed with the aid of the MLFMA octree constructed by placing the object in a cubic box and bisecting each dimension of the cubic box recursively.

The key step in designing an efficient parallelization approach for the MLFMA is partitioning the octree among processes following a certain parallelization approach, the hybrid [7], the hierarchical [20], or the ternary [25]. Compare with the former two strategies, the ternary parallelization approach is proved to be flexible in terms of the number of processes and be suitable for large-scale parallelization. In [25], it was used to solve problems with over ten billion unknowns, which was the largest number of unknowns to be computed via MLFMA at that time.

In the GPU-PMLFMA proposed in this paper, we still use the ternary parallelization approach to partition the MLFMA octree among MPI processes [25]. In the ternary parallelization approach, the MLFMA tree is grouped into plane wave partitioning (PWP) levels, hierarchical structure partitioning (HSP) levels and box partitioning (BP) levels from the top to the lowest level. Each partition strategy partitions the workload in a specified manner: On PWP levels, each box partitions all its plane waves evenly on MPI processes along the $\theta$-direction; On BP levels, all plane waves of a given box are partitioned on the same process; On HSP levels, both boxes and plane waves are partitioned simultaneously. To switch patterns between BP and HSP levels, a virtual local transition (LT) level is designed, at which inner group all-to-all communications are required.

To meet the demand of mesh generation involving tens of billions of unknowns, the auxiliary-tree-based parallel mesh refinement technique provided in [25] is used to generate the final meshes. Moreover, the hybrid octree storage strategy is also used to reduce the memory requirement for storing the MLFMA tree involving billions of leaves. Readers can refer to [25] for additional technical details regarding these techniques.
III. GPU ARCHITECTURE AND THE HYBRID PARALLEL PROGRAMMING MODEL

Before giving details of the GPU-PMLFMA, we review the GPU/CUDA architecture, and the hybrid MPI, OpenMP and CUDA programming model using CPU/GPU asynchronous computing pattern.

In CUDA programming model, the task of a kernel is divided up into threads. The threads are further organized into blocks. Fig. 1 shows a typical thread block organization architecture of a CUDA-capable GPU. CUDA devices have several different memory spaces, Global, local, texture, constant, shared, etc. Each type of memory has its advantages and disadvantages. In this paper, only the global memory and the shared memory are specially referred. The global memory of GPU device is similar to the random access memory (RAM) to CPU. It resides off chip and shared by all the SMs in one device. The shared memory is a special kind of on-chip memory that can be much faster than the global memory, but it is only tens of Kbyte in size. The shared memory is visible to all threads within that block and lasts for the duration of the block. Shared memory provides a convenient way for the threads in the same block to communicate and cooperate with each other. Therefore, the full utilization of the hierarchical memory and reduction of data communications are crucial in GPU computation.

MPI, OpenMP and CUDA are three different parallel programming models targeted for different uses. The MPI is a standardized and portable message-passing standard designed to help user managing a parallel computation on a distributed memory system, while the OpenMP programming model is targeted toward use on shared memory multi-core CPU system and the CUDA is developed for general computing on GPU. Although the theoretical performance of a CPU is much lower than a GPU accelerator, the effective computing power of a multi-core CPU should not be ignored, especially for these cases where frequent memory access are required. The processing on the GPU can be performed asynchronously to the host CPU, with the running time for the former overlapped with the latter to further reduce the wall-clock running time.

IV. GPU ACCELERATION IMPLEMENTATION OF THE TERNARY PMLFMA

Using the CPU and GPU asynchronous computing pattern, a typical hybrid MPI, OpenMP and CUDA parallel programming model is shown in Fig. 2. The serial phase of the program on each MPI process is first executed by the master thread on the host (CPU). Then, in the parallel phase, two independent tasks, or two subtasks generated from decomposing the same computation-intensive task according to the workload, are taken over by GPU and CPU respectively. Each MPI process manages one GPU device, which is used to execute the parallel kernel functions and a large number of GPU threads are generated and organized into a grid of blocks to exploit data parallelism. Multiple CPU threads are invoked on each MPI process by using OpenMP to accelerate the CPU computation. After every GPU device and the corresponding CPU threads finishing their parallel work, the host process will pick up the run time and execute the instructions in the serial phase.

In PMLFMA, the duration of the computation can be divided into two major parts: the setup and the iteratively solution. The computationally intensive parts during the setup procedure of the parallel MLFMA include the calculation of the near-field system matrix, and the calculation of the far-field system matrices, such as the aggregation/disaggregation matrix, and the calculation of translation operators. Generally, the computation time for the near-field matrix filling is larger than that for the far-field matrix filling. In the iteratively solution procedure, the computation intensive parts can also be separated into two parts: the near field interaction, which is essentially a sparse matrix-vector multiplication (SpMV); the far field interaction, which contains three stages, namely the aggregation, the translation, and the disaggregation. Although memory for storing the near field matrix is far larger than the far filed matrix system mentioned above, the duration of the computation for the latter part is far larger than the former. Being aware of this fact, the execution order of different stages of the PMLFMA is carefully designed, as shown in Fig. 3, in which the hybrid MPI, OpenMP and CUDA programming model proposed in Fig. 2 is used. In the following parts, the main stages of the approach are described in detail.
A. Setup Stage

As shown in Fig.3, at the very beginning, some preparation tasks are executed by using only CPU, including the auxiliary tree based parallel mesh refinement, the constructing of hybrid stored MLFMA-octree and the ternary partitioning of the MLFMA octree. After the preparation tasks are completed, the near-field matrix filling is done with the aid of GPU. We decompose the total workload of near-interaction matrix filling into two parts according to the computing power of CPU and GPU. The larger part is calculated on GPU utilizing CUDA programming model, and the resulting data is transferred back to CPU host memory to store it as long as the calculation is finished. At the same time, the smaller part is calculated by CPU using OpenMP multi-threading programming model, and stored directly into its corresponding position. The optimal workload distribution of CPU and GPU in the setup stage can be determined according to a few pre-tested examples running on the same computer platform. According to our numerical experience, for the platform used in the numerical results section V, 95% GPU and 5% CPU workload distribution in the setup stage is acceptable. With this distribution, the computation time cost for these two parts can be overlapped by choosing a proper workload distribution ratio between CPU and GPU.

When an electrically large problem with over tens of billions of unknowns is simulated, since there are enough boxes at the lowest level, the idea of ‘one thread per near box pair’ and ‘several thread warps per block’ is used, as illustrated in Fig.4. Each near box pair is assigned to a GPU thread, then a set of threads with the total number equals to several times of the GPU warp size are organized as a block. During the execution, each thread first fetches the required mesh data for each RWG basis associated with an edge in a near box pair in the global memory, then calculates corresponding nonzero entries in near-field matrix following the procedure of MoM. With this strategy, a high computational throughput for the GPU calculation can be well promised, regardless of the geometrical shape of the object. The calculated larger part of near-field matrix is then transferred back to the CPU host memory and stored together with those computed by CPU. After that, all the GPU global memory occupied is freed.

Then the geometry data required for far field matrices filling is sent to the global memory of GPU. After finishing data transformation between CPU and GPU, far field related matrices, include the aggregation and disaggregation matrices, the translation operators, are filled using GPU. A pre-allocated chunk of memory is used on the global memory of GPU to store these far field matrices of MLFMA. It is persistent between kernel launches, which can reduce the data transformation between GPU and CPU during iterative solution procedure, thereby significantly improve the computational efficiency and benefit data re-using. At the same time, the CPU calculates the right-hand side vector and fills these auxiliary arrays to be used for MPI communication in far field interaction computing during the iteratively solution procedure. Part of these auxiliary arrays to be used in far field interaction calculation are also transferred to GPU global memory.

Similar to the parallelization strategy of near field matrix filling illustrated in Fig. 4, the basic idea for the parallelization of aggregation and disaggregation matrices filling is ‘one thread per box’ and ‘several thread warps per block’. The whole workload of filling aggregation and disaggregation matrices belonging to a box is assigned to a thread, and then one or more thread wraps are organized into a block. For the filling of translation operators, the idea of ‘one thread per plane wave’ and ‘one block per operator’ is used. In the parallel implementation, each translation operator is assigned to a block of threads with size of the thread block set as the number of plane waves belonging to the translation operator, and then each plane wave is assigned to a thread, as illustrated in Fig. 5.
straightforward. In the following, we present the key designs of acceleration of SpMV for near interaction is simple and programming model. We further subdivided the SpMV of near interaction calculation into several smaller parts and make the data communication overlapped with the computation. Then, the radiated plane waves of the boxes at the lowest level are shifted and child boxes plane waves are combined together to obtain parent level boxes on the higher level. This procedure continues until it reaches the second level of the MLFMA tree. In the ternary parallelization approach, the MLFMA tree levels are divided into BP, HSP and PWP levels from the bottom to the highest level; the operations that are required during the aggregation stage differ among them. In coincidence with the ternary MLFMA tree partitioning strategy, different parallelization strategies are also adopted for GPU at different MLFMA tree levels.

On the BP levels, each process is assigned a large number of boxes, and all the plane waves in each box. For the GPU acceleration of aggregation stage on BP levels, the idea of ‘one thread per box’ and ‘several warps of threads per block’ is used, which is similar to the one illustrated in Fig. 4 except ‘box’ is used instead of ‘box pair’. The reason for choosing this parallel strategy is that, on one hand, there are enough boxes to get a high parallel efficiency, even at the highest BP level when problems involving tens of billions of unknowns are solved. On the other hand, the symmetry of the plane waves is used on the BP levels to halve the memory. Therefore, the plane wave operation in a box is inconsistent. The assignment of plane waves in the same box may bring slowdowns due to warp-divergence caused by multiple flow paths (i.e. if statements). Since each box on the highest BP level and all its descendants are partitioned on the same process, all the plane waves of child boxes that are required for calculating the plane waves of a higher-level box are local. Hence, the aggregation stage on these BP levels does not require any CPU-GPU data communications.

On the HSP levels, both the boxes and plane waves in each box are simultaneous partitioned among a group of processes. For a general 3D object, the number of boxes decreases approximately fourfold, while the number of plane waves in each box increases at approximately the same speed. In most cases, even on the highest HSP level, the number of boxes whose plane waves are partitioned to a process along the θ-direction is much greater than the number of GPU SM. Hence, for HSP levels, the idea of ‘one block per box’ and ‘one thread per planewave’ is used, as illustrated in Fig. 7. We assign each child box to a block and the plane waves in it to a thread. During aggregation on HSP levels, the inter-process communication for preparing the data for interpolation is conducted between two neighboring processes. Therefore, data communication between CPU and GPU is unavoidable. To make the data communication overlapped with the computation,
we divide the calculation into two local and nonlocal two parts, and use the CPU/GPU asynchrony pattern. At the very beginning of the aggregation on a HSP level, each MPI process send the data to its neighbor processes using nonblocking buffered send. Then each process compute the contribution of local aggregation operation using GPU acceleration. During this time, each process finishes receiving data from neighbor processes and then transfers it to GPU host memory to finish nonlocal aggregation procedure. By such design, the CPU-CPU and CPU-GPU communications are overlapped by the local aggregation operation.

![Fig. 6. Parallel scheme for aggregation on HSP levels. (a) The idea of ‘one block per box’. (b) Plane waves in a box is divided into local and nonlocal parts and partitioned using the idea of ‘one thread per plane wave’.

On the PWP levels, the plane waves in each box is partitioned evenly among all MPI processes along the θ-direction. Since the number of boxes on PWP levels is usually very small, the idea of ‘several blocks per box’ and ‘one thread per planewave’ is used. On the lowest PWP level, plane waves of each box are evenly partitioned into four parts at first. Then, each part is assigned to a thread block, with the computation work of each planewave assigned to a thread. When goes to the higher PWP level, the number of blocks assigned to plane waves in a box is increased by fourfold, as illustrated in Fig. 7. In this way, there will be a sufficient number of both blocks and threads for the parallel computation on PWP levels to ensure a high parallel efficiency. Similar to the HSP level, the inter-process communication is conducted between two neighboring processes. Also, the plane waves are divided into local and nonlocal parts, and the CPU/GPU asynchrony pattern is used to make the communication and computation overlapped.

![Fig. 7. Illustration of ‘several blocks per box’ scheme for aggregation on PWP levels. (a) boxes on the current level (b) the parent box on a higher level.

The disaggregation is conducted via pre-order traversal of the MLFMA tree from the second level to the lowest level, which is opposite the traversal direction for aggregation. During disaggregation, the translation of the plane waves belonging to the second near boxes on the same level is done at the same time. The parallelization strategies used for disaggregation and translation on different MLFMA levels are similar as the aggregation, except the observer ‘parent box’ is used instead of ‘son box’ in the disaggregation phase.

At the lowest level, the incoming plane waves are multiplied by the disaggregation matrix and received by the testing functions, with the in-place parallel reduction tree-based algorithm to speed up the array self-summation. The final results of far interaction are then transferred back to CPU host memory and summed with the near interaction results to complete the matrix-vector multiplication.

C. GPU Architecture-based Optimization

To further improve computational efficiency of the proposed GPU-PMLFMA, two effective GPU architecture-based optimization strategies are implemented.

The first one is data re-using by shared memory. One of the key problems in optimizing the GPU-PMLFMA is to optimize the sparse matrix vector multiplication (SpMV). Take the interpolation stage on HSP levels as an example. Since the idea of ‘one block per box’ and ‘one thread per planewave’ is used, the SpMV is performed by column. Each thread is assigned a column of matrix elements to be multiplied by the corresponding plane wave. When goes to the higher PWP level, the number of blocks assigned to plane waves in a box is increased by fourfold, as illustrated in Fig. 7. In this way, there will be a sufficient number of both blocks and threads for the parallel computation on PWP levels to ensure a high parallel efficiency. Similar to the HSP level, the inter-process communication is conducted between two neighboring processes. Also, the plane waves are divided into local and nonlocal parts, and the CPU/GPU asynchrony pattern is used to make the communication and computation overlapped.

![Fig. 8. Illustration of process.

0018-926X (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
The other optimization method is the in-place parallel reduction tree-based algorithm. As stated in the last subsection, during the aggregation/disaggregation, to calculate the radiation/reviving patterns, a sum reduction operation is required. To improve efficiency of the code and benefit parallel computing, the in-place parallel reduction tree-based algorithm is used in this paper, as illustrated in Fig. 9. The algorithm takes \( \log(N) \) steps for the summation of \( N \) elements.

First, we show the parallelization efficiency of this parallel approach. Fig. 10 presents the parallelization efficiency for the total duration (including the setup and iterations) of solving the scattering by a sphere of radius 207 m at 0.3 GHz discretized with 201,392,784 unknowns. The solution is parallelized onto 10, 20, 30, 40, 60, 80, and 100 nodes, with 4 MPI processes and 4 accelerators per node. In this figure, the parallel efficiency when 10 nodes are used is defined as 100%. The parallel efficiency remains as high as 84% on 40 nodes and drops to about 60% on 100 nodes. The parallel efficiency of GPU-PMLFMA is lower than the PMLFMA. This is reasonable, because with the increment of computing nodes, the computation time becomes smaller and the MPI communication time, which cannot be reduced by using more computer nodes, becomes more important.

Then, we validate the proposed approach by comparing the Radar Cross Sections (RCSs) of a conducting sphere with Mie series. The diameter of the sphere is 3040m, and illuminated by a 0.3 GHz plane wave. After four rounds of uniform refinement and two additional rounds of parallel uniform refinement, the sphere is discretized into 6,712,503,584 curvilinear triangular patches with 10,068,755,376 unknowns. The GMRES solver converges to 0.001 in 158 iterations. It can be observed in Fig 11. that the PMLFMA, the GPU-PMLFMA and Mie series show a very good agreement.
In order to demonstrate the efficiency of the GPU-PMLFMA, the RCS of conducting spheres with diameters of 100λ, 200λ, 400λ, 800λ, and 1600λ are calculated by using 2, 8, 32, 128, and 512 MPI processes respectively. The speedup of different stages in MLFMA calculation are shown in Fig. 12. As can be seen from this figure, the acceleration of near-field matrix assembly is excellent, which increases as the number of unknowns grows. The acceleration of translation matrix assembly stage shows the same trend. The acceleration of the aggregation/disaggregation matrix assembly stages is more significant and varies dramatically. This because the total CPU time spent on the aggregation/disaggregation matrix assembling procedure is very short, and a small difference of a few tenths of a second level makes a great difference on the acceleration. When the unknown number increases, the acceleration in the GMRES solution increases slowly. This is because when the number of unknowns becomes larger, the computation time becomes more important. Therefore, the computational capability of GPU for compute-intensive parts is better explored. Note the speedup of GMRES iterative solution is restricted not only by the data communications between the host and device, but also by the data communications between MPI processes. Therefore, it is more difficult to get as high a speedup as those matrix filling parts.

The computational statistics for the sphere with diameter 1600λ. are presented in Table I, in which ‘CPU’ and ‘GPU’ represent the PMLFMA and the GPU-PMLFMA, respectively. Compared with the eight-threaded PMLFMA, the speedup of GPU-PMLFMA is 3.25, almost the same as the speedup of GMRES iterative solution. This is also true for other electrically large problems involving tens of billions of unknowns.

### Table I Speedup of Bistatic RCS Calculation of a Conducting Sphere With Diameter of 1600λ.

|       | PMLFMA (sec.) | GPU-PMLFMA (sec.) | Speedup |
|-------|---------------|-------------------|---------|
| Vr and Vf | 3.88          | 0.71              | 5.46    |
| T      | 271.7         | 63.15             | 4.30    |
| Znear  | 1390          | 215.6             | 6.45    |
| GMRES  | 11724         | 3840              | 3.05    |
| Total Time | 13390        | 4119              | 3.25    |

To further illustrate the capability and efficiency of the proposed method, a ship model is considered. The ship model with a length of 180m is illuminated by plane waves with frequencies of 0.5 GHz, 1 GHz, 2 GHz, 4 GHz, and 10 GHz, and these examples are calculated by using 2, 8, 32, 128, and 512 processes respectively. The speedup versus the number of unknowns is shown in Fig. 13. In Fig 13, similar speedups with those for the conducting sphere can be observed. Comparing Table I and Table II, we can know if the large problem has a similar number of unknowns, the speedup for each part is similar. In other words, the algorithm has strong stability and it is insensitive to the geometrical shape of the object.

### Table II Speedup of Bistatic RCS Calculation of a Ship Model at 10GHz.

|       | PMLFMA (sec.) | GPU-PMLFMA (sec.) | Speedup |
|-------|---------------|-------------------|---------|
| Vr and Vf | 6.51          | 0.24              | 27.13   |
| T      | 1964          | 169.0             | 11.62   |
| Znear  | 1225          | 170.1             | 7.21    |
| GMRES  | 14022         | 4759              | 2.95    |
| Total Time | 17218        | 5095              | 3.38    |
At last, to show superior performance of the proposed GPU-PMLFMA, two electrical large objects discretized with tens of billions of unknowns are simulated. The first object is an aircraft has a length of 57.5 m, as illustrated in Fig. 14. Fig. 15 presents the VV-polarized bistatic RCS at 90 GHz. At this frequency, the aircraft has a maximum dimension of 17254λ. In this calculation, a final mesh with 21,814,265,088 unknowns was used. After 119 iterations, a relative residual error of 0.005 was realized. The duration of the computation was 6 hours and 51 minutes. A total of 5120 GPU accelerators were used in this calculation. In total, 74.7 TByte RAM on CPU and 81.9 TByte on GPU were used. Additional simulation details are presented in Table III.

The second model is a ship model has a length of 179.4 m, as shown in Fig. 16. Fig. 17 presents the VV-polarized bistatic RCS of the ship at 40 GHz. At this frequency, it has a maximum dimension of 23940λ. In this calculation, a final mesh with 41,839,886,016 unknowns was used. This is the largest problem scale of parallel MLFMA computability reported in the CEM area to date. After 112 iterations, a relative residual error of 0.005 was realized. The duration of the computation was 8 hours and 31 minutes. A total of 5120 GPUs are used in this calculation. This is also the largest implementation of heterogeneous computing in CEM field. In total, 187 TByte RAM on CPU and 163 TByte on GPU were used. Detailed information is provided in Table III.

| Target         | Sphere                  | Aircraft model          | Ship model               |
|----------------|-------------------------|-------------------------|--------------------------|
| Maximums dimension | 3042λ                  | 17254λ                  | 23940λ                   |
| Unknowns       | 10,068,755,376          | 21,814,265,088          | 41,839,886,016           |
| MLFMA levels   | 14                      | 16                      | 16                       |
| HSP levels     | 3–10                    | 4–10                    | 4–11                     |
| Number of processes | 2560                  | 5120                    | 10240                    |
| Number of CPU cores | 2560*8                | 5120*8                  | 10240*8                  |
| Number of GPUs  | 2560                    | 5120                    | 10240                    |
| Matrix filling time | 597s                   | 1328s                   | 1906s                    |
| Residual error  | 0.001                   | 0.005                   | 0.005                    |
| Iteration number | 158                    | 119                     | 112                      |
| Iteration time  | 3h, 20min               | 6h, 20min               | 7h, 49min                |
| Total time      | 3h, 39min               | 6h, 51min               | 8h, 31min                |
| Peak memory on CPU | 42.8TB                  | 74.7TB                  | 187TB                    |
| Peak memory on all GPUs | 41TB                    | 81.9TB                  | 163TB                    |

Table III: Simulation Details

Fig. 14. Geometric illustration of an aircraft model.

Fig. 15. VV polarized bistatic RCS of the aircraft model in Fig. 13 at 90 GHz. The target is illuminated by a plane wave that is propagating in the xy-plane at 30 degrees from the x-axis.

Fig. 16. Geometric illustration of a ship model.

Fig. 17. VV and VH polarized bistatic RCSs of the ship model in Fig. 16 at 40 GHz. The target is illuminated by a plane wave that is propagating in the yz-plane at 70 degrees from the +z-axis. The observation plane is defined by θ=70° and ϕ=0–360°.
VI. CONCLUSION

In this paper, we present a hybrid MPI, OpenMP and CUDA based parallel approach of MLFMA for extremely large 3D scattering problems. For parallelization on MPI processes, the flexible and efficient ternary partitioning scheme is employed, while for parallelization on a single GPU, different parallelization strategies are used at different MLFMA tree levels to achieve a good computation performance. Special memory usage strategy is designed with both the CPU memory and the GPU device memory to benefit data re-using as well. The CPU/GPU asynchronous computing pattern is designed with the OpenMP and CUDA for accelerating corresponding calculation respectively. Further GPU optimization strategies are implemented to improve the computational efficiency. The capability and efficiency of the proposed approach are demonstrated by solving scattering large problems. Numerical results show the total speedup of the GPU-MLFMA is over 3 times compared with eight-threaded PMLFMA. We present the RCS from an extremely ship model and an aircraft model with more than tens of billions of unknowns. The maximum dimension of the ship reaches up to about 24,000λ, and the number of unknowns is larger than 41.8 billion.

REFERENCES

[1] R. Coifman, V. Rokhlin, and S. Wandzura, “The fast multipole method for the wave equation: a pedestrian prescription,” IEEE Antennas Propag. Mag., vol. 35, no. 3, pp. 7-12, Jun. 1993.
[2] J. M. Song, C. C. Lu, and W. C. Chew, “Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects,” IEEE Trans. Antennas Propag., vol. 45, no. 10, 1488-1493, Oct. 1997.
[3] X. Q. Sheng, J. M. Jin, J. Song, W. C. Chew, and C. C. Lu, “Solution of combined-field integral equation using multilevel fast multipole algorithm for scattering by homogeneous bodies,” IEEE Trans. Antennas Propag., vol. 46, no. 11, pp. 1718-1726, Nov. 1998.
[4] M. F. Catedra, E. Gago and L. Nuno, “A numerical scheme to obtain the RCS of three-dimensional bodies of size using the conjugate gradient method and the fast Fourier transform,” IEEE Trans. Antennas Propagat., vol. 37, pp. 528-537, May 1989.
[5] F. Wu, Y. Zhang, Z. Z. Ou, and E. Li, “Parallel multilevel fast multipole method for solving large-scale problems,” IEEE Antennas Propag. Mag., vol. 47, no. 4, pp. 110-118, Aug. 2005.
[6] C. Waltz, K. Sertel, M. A. Carr, B. C. Usner, and J. L. Volakis, “Massively parallel fast multipole method solution of large electromagnetic scattering problems,” IEEE Trans. Antennas Propag., vol. 55, no. 6, pp. 1810-1816, Jun. 2007.
[7] S. Velamparibamb, W. C. Chew, and J. M. Song, “10 million unknowns: Is it that big?” IEEE Antennas Propag. Mag., vol. 45, no. 2, pp. 43-58, Apr. 2003.
[8] S. Velamparibamb and W. C. Chew, “Analysis and performance of a distributed memory multilevel fast multipole algorithm,” IEEE Trans. Antennas Propag., vol. 53, no. 8, pp. 2719-2727, Aug. 2005.
[9] O. Ergul and O. Gurel, “Fast and accurate solutions of extremely large integral-equation problems discretised with tens of millions of unknowns,” Electron. Lett., vol. 43, no. 9, pp. 499-500, Apr. 2007.
[10] J. Fostier and F. Olyslager, “An asynchronous parallel MLFMA for scattering at multiple dielectric objects,” IEEE Trans. Antennas Propag., vol. 56, no. 8, pp. 2346-2355, Aug. 2008.
[11] O. Ergul and L. Gurel, “Efficient parallelization of the multilevel fast multipole algorithm for the solution of large-scale scattering problems,” IEEE Trans. Antennas Propag., vol. 56, no. 8, pp. 2355-2365, Aug. 2008.
[12] O. Ergul and X. Q. Sheng, “A sophisticated parallel MLFMA for scattering by extremely large targets,” IEEE Antennas Propag. Mag., vol. 50, no. 3, pp. 129-138, Jun. 2008.
[13] X. M. Pan, W. C. Pi, M. L. Yang, Z. Peng, and X. Q. Sheng, “Solving problems with over one billion unknowns by the MLFMA,” IEEE Trans. Antennas Propag., vol. 57, no. 5, pp. 1297-1305, May 2009.
[14] J. Hu, Z. P. Nie, L. Lei, J. Hu, X. D. Gong, and H. P. Zhao, “Fast 3D EM scattering and radiation solvers based on MLFMA,” J. Syst. Eng. Electron., vol. 19, no. 2, pp. 252-258, Apr. 2008.
[15] M. G. Araujo, J. M. Taboada, F. Obelleiro, J. M. Bertolo, L. Landesa, J. Rivero, and J. L. Rodriguez, “Supercomputer aware approach for the solution of challenging electromagnetic problems,” Progr. Electromagn. Res., vol. 101, pp. 241-256, 2010.
[16] J. M. Taboada, M. G. Araujo, J. M. Bertolo, L. Landesa, F. Obelleiro, and J. L. Rodriguez, “MLFMA-FFT parallel algorithm for the solution of large-scale problems in electromagnetics,” Progr. Electromagn. Res., vol. 105, pp. 15-30, 2010.
[17] J. M. Taboada, M. G. Araujo, F. Obelleiro, J. L. Rodriguez, and L. Landesa, “MLFMA-FFT parallel algorithm for the solution of extremely large problems in electromagnetics,” Proc. IEEE, vol. 101, no. 2, pp. 350-363, Feb. 2013.
[18] V. Melapudi, B. Shanker, S. Seal, and S. Aluru, “A scalable parallel wideband mlfma for efficient electromagnetic simulations on large scale clusters,” IEEE Trans. Antennas Propag., vol. 59, no. 7, pp. 2565-2577, 2011.
[19] B. MacKie-Mason, Y. Shao, A. Greenwood and Z. Peng, “Supercomputing-Enabled First-Principles Analysis of Radio Wave Propagation in Urban Environments,” IEEE Trans. Antennas Propag., vol. 66, no. 12, pp. 6606-6617, Dec. 2018.
[20] O. Ergul and L. Gurel, “Hierarchical parallelisation strategy for multilevel fast multipole algorithm in computational electromagnetics,” Electron. Lett., vol. 44, no. 6, pp. 3-4, 2008.
[21] O. Ergul and L. Gurel, “A hierarchical partitioning strategy for an efficient parallelization of the multilevel fast multipole algorithm,” IEEE Trans. Antennas Propag., vol. 57, no. 6, pp. 1740-1750, Jun. 2009.
[22] O. Ergul and L. Gurel, “Rigorous solutions of electromagnetics problems involving hundreds of millions of unknowns,” IEEE Antennas Propag. Mag., vol. 53, no. 1, pp. 18-27, Feb. 2011.
[23] O. Ergul and L. Gurel, “Hierarchical parallelization of the multilevel fast multipole algorithm (MLFMA),” Proc. IEEE, vol. 101, no. 2, pp. 332-341, 2013.
[24] B. Michiels, J. Fostier, I. Bogaert, and D. D. Zutter, “Full-Wave simulations of electromagnetic scattering problems with billions of unknowns,” IEEE Trans. Antennas Propag., vol. 57, no. 6, pp. 796-798, Feb. 2015.
[25] M. L. Yang, B. Y. Wu, H. W. Gao, and X. Q. Sheng, “A ternary parallelization approach of MLFMA for solving electromagnetic scattering problems with over 10 billion unknowns,” IEEE Trans. Antennas Propag., vol. 67, no. 11, pp. 6965-6979, 2019.
[26] M. Abduljabbar, M. A. Farhan, N. Al-Harthi, R. Chen, Y. Yokota, H. Bagi, and D. keyes, “Extreme Scale FMM-Accelerated Boundary Integral Equation Solver for Wave Scattering,” SIAM Journal on Scientific Computing, vol. 41, no. 3, pp. C245-C268, 2018.
[27] M. P. Lingg, S. M. Hughey, H. M. Akutga, and B. Shanker, “High performance evaluation of helmholtz potentials using the multi-level fast multipole algorithm,” IEEE Trans. Antennas Propag., vol. 67, no. 11, pp. 6965-6979, 2019.
[28] M. Cwikla, J. Aronsson, and V. Okhmatovski, “Low-frequency MLFMA on graphics processors,” IEEE Antennas Wireless Propag. Lett., vol. 9, pp. 8-11, 2010.
[29] J. Guan, Y. Su, 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, 2013.
[30] N. Tran, and O. Kille, “Parallel implementations of multilevel fast multipole algorithm on graphical processing unit cluster for large-scale electromagnetics objects,” Appl Comput Electromag Soc J, vol. 1, no. 4, pp. 145-148, 2016.
[31] T. Phan, N. Tran, and O. Kille, “Multi-level fast multipole algorithm for 3-D homogeneous dielectric objects using MPI-CUDA on GPU cluster,” Appl Comput Electromag Soc J, vol. 3, no. 3, pp. 335-338, 2018.
[32] V. Fraysse, L. Giraud, S. Gratton, and J. Langou, “Algorithm 842: A set of GMRES routines for real and complex arithmetics on high performance computers,” ACM Trans. Math. Softw., vol. 31, pp. 228-238, Jun. 2005.
Wei-Jia He received the B.S. degree in electronic information engineering from the Beijing Institute of Technology, Beijing, China, in 2017, where he is currently pursuing the Ph.D. degree in electrical engineering with the Institute for Applied Electromagnetics, School of Integrated circuits and Electronics.

His current research interests include high-performance parallel computational techniques and heterogeneous parallel computing.

Zeng Yang received the B.S. degree in electronic information engineering from the Hubei University, Wuhan, China, in 2016 and the M.S. degree from the Hangzhou Dianzi University, Hangzhou, in 2019. He is currently pursuing the Ph.D. degree in electrical engineering with the Institute for Applied Electromagnetics, School of Integrated circuits and Electronics, Beijing Institute of Technology, Beijing, China.

His current research interests include domain decomposition methods, and applied computational electromagnetics to solve large-scale and multiscale scattering and radiation problems.

Xiao-Wei Huang received the B.S. degree in electronic information engineering from the Beijing Institute of Technology, Beijing, China, in 2017, where he is currently pursuing the Ph.D. degree in electrical engineering with the Institute for Applied Electromagnetics, School of Integrated circuits and Electronics. His current research interests include computational electromagnetics, domain decomposition methods, and fast direct solvers for integral equations.

Mr. Huang is a recipient of the Best Student Paper Award (1st place) at the 2018 International Applied Computational Electromagnetics Society (ACES-China) Symposium.

Wu Wang received the B.S. degree from School of Mathematics, Peking University in Beijing 2004 and Ph. D. degree from Graduate University of Chinese Academy of Sciences in Beijing 2010, and did a postdoctoral work in Graduate School of Informatics, Koyto University, in Japan, 2011-2012.

He is currently an Associate Professor with Computer Network Information Centre, Chinese Academy of Sciences in Beijing.

He has authored and co-authored over 30 journal and conference papers. His current research interests include fast algorithms for electromagnetics, high-performance parallel computing and their applications in the field of electromagnetic scattering, cosmological N-body problem.

Ming-Lin Yang received the B.S. and Ph.D. degrees from the Beijing Institute of Technology (BIT), Beijing, China, in 2007 and 2015, respectively, and the Ph.D. degree from the University of Rouen, Rouen, France, in 2014.

He is currently an Associate Professor with the Institute of Applied Electromagnetics, School of Information and Electronics, Beijing Institute of Technology. He has authored and co-authored over 30 journal papers and 20 conference papers. His current research interests include fast multipole and domain decomposition methods based fast algorithms for 3D objects with complex shape and materials, high-performance parallel computational techniques and their applications in the field of electromagnetic scattering/antenna radiation/ light interactions with particles.

Prof. Yang was a recipient of the scholarship award for Excellent Doctoral Student granted by Ministry of Education in 2010, the nomination award of Excellent PhD thesis of Chinese Institute of Electronics in 2016, the IEEE Antennas and Propagation Society Ulrich L. Rohde Innovative Conference Paper Awards on Computational Techniques in Electromagnetics in 2018, the Young Scientist Award of PIERS in 2019.

Xin-Qing Sheng is a Chair Professor of the Beijing Institute of Technology. Prof. Sheng has authored and coauthored over 150 papers in referred journals, and seven books. Prof. Sheng has authored the software of SINOCOM for electromagnetic simulation. His research interests include computational electromagnetics, scattering and antenna analysis, electromagnetic compatibility, microwave imaging.

Prof. Sheng is a recipient of the 2001 One Hundred Talents Program awarded by the Chinese Academy of Sciences, the 2004 Cheung Kong Scholar Program awarded by the Ministry of Education of China, and the first recipient of the first prize of Beijing Science and Technology Awards in 2009.