Parallel Higher-Order Method of Moments with Efficient Out-of-GPU Memory Schemes for Solving Electromagnetic Problems

Zhongchao Lin, Yan Chen, Xunwang Zhao*, Daniel Garcia-Donoro, Yu Zhang, and Huanhuan Zhang

School of Electronic Engineering
Xidian University, Xi’an, Shaanxi 710071, China
xwzhao@mail.xidian.edu.cn

Abstract **─** A distributed parallel Higher-order Method of Moments (HoMoM) for solving electromagnetic problems on CPU/GPU clusters is presented. An MPI/OpenMP/CUDA parallel framework based on the GPU context technique is designed. An out-of-GPU memory scheme is employed to break the limitation of the GPU memory. To improve the performance of data transferring between main memory and GPU memory, an overlapping scheme based on asynchronous technique and CUDA streams is adopted. In comparison with the parallel CPU version only, numerical results including a metallic airplane and an airborne array with dielectric structures demonstrate the high performance of the proposed method.

Index Terms **─** GPU-based HoMoM, GPU context, out-of-GPU memory, overlapping, parallel framework.

I. INTRODUCTION

In the field of computational electromagnetics (CEM), the method of moments (MoM) is widely used for solving electromagnetic radiation and scattering problems [1]. It is well known that direct lower/upper (LU) decomposition solvers and iterative solvers are the most common ways to solve the matrix equations of MoM. In order to avoid slow convergence rates or divergence issues of iterative solvers, direct LU decomposition algorithms are utilized as matrix equation solvers. Unfortunately, with the electrical size of problems increasing, the solution time of LU solver increases rapidly due to the computational complexity of $O(N^3)$, where N is the number of unknowns. With the rapid development in computer hardware capabilities, parallel computing technique has been an efficiently approach for solving extremely complicated engineering problems. In recent years, the Graphics Processing Unit (GPU) has become a prevalent commodity in parallel computing due to its powerful computational capability.

Since the Nvidia GPUs programmed through the CUDA API was introduced in 2006 [2], the GPUs provide a very attractive, low-cost hardware platform for CEM. The application of GPU in the area of CEM started in the finite-difference time-domain method (FDTD). In [3-5], the GPU-accelerated FDTD was implemented to deal with the 2D and 3D simulation problems, a good acceleration ratio can be obtained. However, MoM has received relatively little attention in the GPU context. The application of GPU acceleration using CUDA to MoM is presented in [6-8], and an iterative solution scheme for the linear system was adopted rather than a direct solving scheme. In [9-11], a GPU-accelerated implementation using Nvidia CUDA for the matrix assembly of the MoM using Rao-Wilton-Glisson (RWG) basis functions [12] was presented. However, the utilization of MAGMA library [13] prohibits its application on a distributed memory platform. In [14], an out-of-core scalable approach that can break the restrictions of GPU memory was introduced, but its performance get worse without the optimization of data-movement between GPU and CPU. In [15], an approach integrating the CUDA computing directly into the ScaLAPACK framework was presented and good speedup was obtained. However, the scale of matrix can be factorized is limited by the GPU memory. In very recent papers by Topa [16] and Mu [17], some efficient out-of-core techniques of GPU-accelerated MoM were presented, but these work are also developed on a single CPU/GPU computing platform.

Under this situation, a hybrid parallel CPU/GPU version of a higher-order method of moments (HoMoM) is presented in this paper. The proposed technique makes use of procedures with efficient out-of-GPU memory schemes and able to run on distributed memory systems with multiple CPU/GPU computing nodes. In the particular case of this paper, the GPU is used to accelerate the calculations of the LU decomposition during the matrix factorization step. The scattering of an airplane and the radiation of an airborne array are simulated to demonstrate the acceleration performance of the proposed algorithm. The implementation of the proposed hybrid CPU/GPU technique is summarized as: 1) an efficient MPI/OpenMP/CUDA parallel framework based on GPU

Submitted On: October 31, 2016
Accepted On: August 26, 2017

context technique is adopted to implement the hybrid parallel CPU/GPU procedures; 2) an efficient out-of-GPU memory scheme is utilized to break the limitation of GPU memory, thus offering a possibility of handling complex EM problems; and 3) the asynchronous data transfer and CUDA streams techniques are used to overlap the data-movement and computation, which can effectively avoid the time of data transfer between CPU and GPU. Details about all these procedures are given in Section III.

II. PARALLEL HIGHER-ORDER METHOD OF MOMENTS

A brief review regarding the basic principles of the integral equation theory, higher-order basis functions and the LU decomposition algorithm is given in this section. Readers are referred to [1] for an in-depth discussion of the theory.

A. Integral equations

The electromagnetic theory employed in this paper is based on the so-called Surface Integral Equations (SIEs) [18] in the frequency domain for equivalent electric and magnetic currents over dielectric boundary surfaces and electric currents over Perfect Electric Conductors (PECs). The set of integral equations obtained are solved by using MoM, and specifically using the Galerkin's method. The code is able to handle inhomogeneous dielectrics categorized by a combination of various homogeneous dielectrics. Therefore, any composite metallic and dielectric structure can be represented as an electromagnetic system consisting of a finite number of finite-size linear, homogeneous and isotropic regions situated in an unbounded linear, homogeneous and isotropic environment.

For general models, the integral equation employed by the code is the well-known Poggio-Miller-Chang-Harrington-Wu (PMCHW) formulation [1, 19]. However, when one of the boundary surfaces between two different regions is PEC, the magnetic currents are equal to zero at the boundary surface and that equation degenerates into the electric field integral equation (EFIE) [20].

B. Higher-order basis functions

In order to approximate the solution to the aforementioned integral equation, higher-order polynomials over wires and quadrilateral patches are used as basis functions over relatively large subdomains [1]. Typically, the number of unknowns for the HOBs is reduced by a factor of 10 compared with that for RWGs, and thus the use of HOBs significantly reduces the computational complexity and memory requirement.

There are also some other advantages in using the polynomial basis functions. For example, the intermediate results obtained in evaluating the elements of the impedance matrix for lower-order can be used in the computation of the elements of the impedance matrix when using higher-order polynomials. In addition, Green's function for each pair of integration points belonging to two patches is only evaluated once. These advantages improve the efficiency of the matrix filling for the HOBs presenting a straightforward implementation.

C. LU decomposition algorithm

Once the system of equations is obtained, the code makes use of the LU decomposition algorithm to solve the problem and obtain the solution. Specifically, the code uses the LU right-looking algorithm. This decomposition technique mainly includes the pivoting step, the panel column factorization, the panel row update and, finally, the trailing submatrix update. Given deeper details about the code involved on the LU decomposition, the routines pzgetrf2, pztrsm and pzgemm are the ones responsible of each of the steps. It is worth noting that the update operations contribute more than 80% of the computation time for a large scale dense complex matrix.

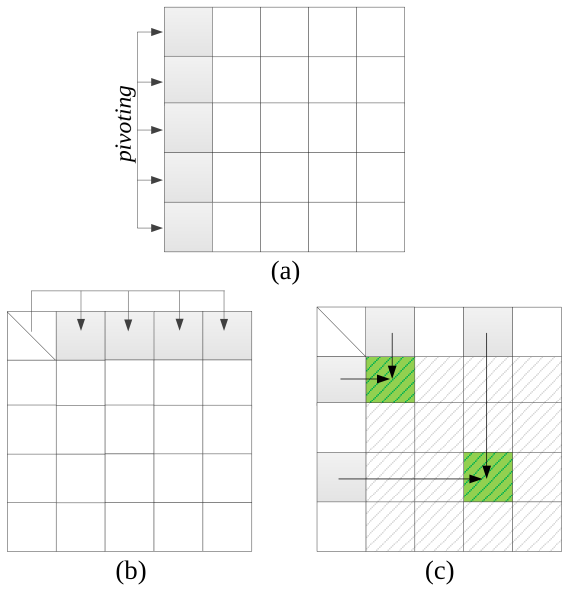


Fig. 1. Processes of LU Decomposition in ScaLAPACK or Intel MKL: (a) panel column factorization, (b) panel row update, and (c) trailing update.

Figure 1 shows a summary of the algorithm where the arrows indicate the data dependency on each step. In this way, for example, during the pivoting step, all the coefficients of the panel column have to be known by all the MPI processes involved on the decomposition. Then, network communication is required between these processes degrading the parallel performance. This behavior can be extrapolated to the rest decomposition steps which are repeatedly executed until the factorization is completed.

III. HYBRID PARALLEL CPU/GPU IMPLEMENTATION

Details about the implementation of the proposed hybrid parallel CPU/GPU technique are given in this section. The parallel implementation of the method is described next meanwhile the out-of-GPU memory scheme is detailed later. It is worth to mention that, only using an MPI-based multi-node processing is not enough to achieve good parallel performance. It is also required an optimization on the data-movement between GPU and CPU memory. All these details are described in the next subsections.

A. Parallel framework based on GPU context

As any other computational technique running on distributed memory CPU/GPU clusters, the proposed GPU HoMoM implementation makes use of MPI to perform the internode communication.

The simplest parallel framework one can consider is to assign one CPU core and one GPU card to each MPI process. However, typically, the number of GPU cards is usually less than that of CPU cores available in the system. Under this scenario, there is an unmatched situation between MPI processes and GPU cards, which would lead to an unbalanced computing power in the different MPI processes. In order to alleviate this issue, different techniques reduce the number of MPI processes of each node to match with the number of GPU cards, meanwhile, multi-threads techniques (*i.e.*, OpenMP) are adopted to make full use of the CPU cores of each node. This improved scheme assigns multiple CPU cores and one GPU card to each MPI process, ensuring a good balance between the computes nodes. However, due to the low number of MPI processes involved on the execution (typically, CPU/GPU cluster has only one GPU card per compute node), the amount of communication needed increases rapidly reducing the performances of the implementation. Therefore, the implementation of an efficient parallel framework with good balance and performance is not straightforward. Fortunately, the context technique of CUDA makes this possible.

Based on the CUDA context technique, multiple MPI processes can use a GPU card simultaneously. Each MPI process opens a CUDA context on the GPU card, and the resources of the GPU card are averagely distributed to each MPI process. It is equivalent to partition one GPU card into several virtual GPU cards. Each MPI process can use its own virtual GPU resources to accelerate the computing tasks. Note that the OpenMP technique can also be adopted in this framework and the communication between CPU cores and GPU context is implemented by PCI-E system bus. The efficient parallel framework based on GPU context is shown in Fig. 2.

It is worth noting that, in systems where only few CPU cores are available per node, the number of virtual GPU card can be equal to the number of CPU cores. In this case, the OpenMP technique is not required to provide a good power balance. However, when the number of CPU cores available per node is larger, the GPU context technique will consume a large amount of GPU resources. Thus, the total performance of the implementation will be drastically degraded. Then, the number of virtual GPU card must be reduced and the OpenMPI technique used.

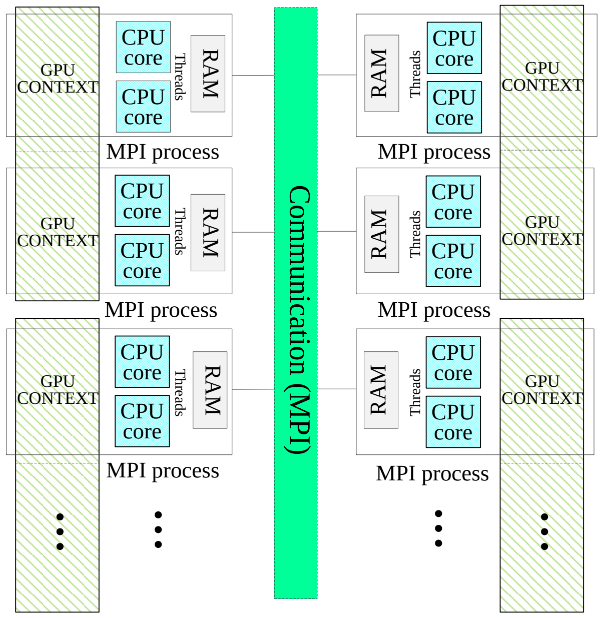


Fig. 2. MPI/OpenMP/CUDA parallel framework base on GPU context technique.

B. Out-of-GPU memory scheme

As mentioned previously, the trailing update operation in the LU decomposition algorithm contributes in more than 80% of the computation time for a large scale dense complex matrix. The, it seems appropriate to accelerate the pzgemm routine employing the GPU power. In this phase, the computing task of each process is reduced to perform matrix multiplication in the form: C=C-AB (see Fig. 3 (a)). The matrix C is located on the process that executes the matrix multiplication, while the matrices A and B are obtained through MPI communication.

The data required for GPU to complete certain compute tasks should be uploaded to it, so the size of the uploaded data has influence on the performances. Thus, in order to accelerate the whole matrix multiplication, the matrices B and C are divided into two parts C_1 and C_2, and B_1 and B_2, respectively. Then, the operation C_1=C_1-AB_1 is performed in the CPU cores, meanwhile the operation C_2=C_2-AB_2 is performed in GPU cards. Fig. 3 (b) illustrates this process.

Note that when matrices B_2 and C_2 are too large to fit in GPU memory, the previous scheme must be improved and this memory limitation has to be broken. Thus, an out-of-GPU memory scheme is implemented where matrices B_2 and C_2 are split into smaller matrices that can fit into the GPU memory. Then, through multiple data transfer and calculation, the process of trailing update is completed. Figure 3 (c) shows a sketch of this out-of-GPU scheme.

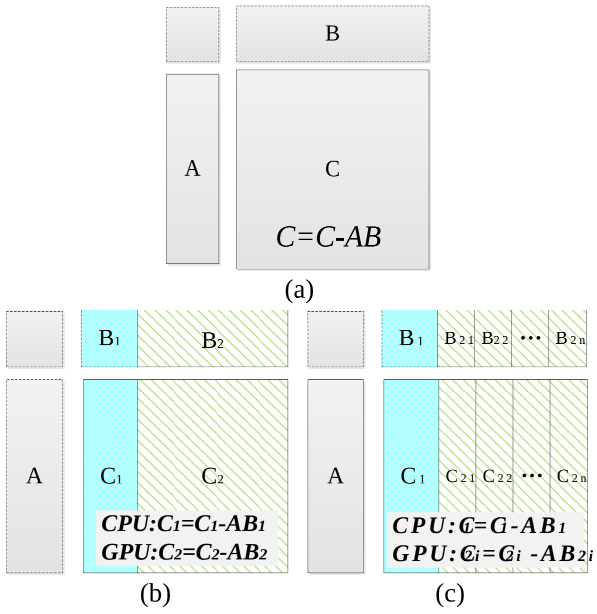


Fig. 3. The out-of-GPU memory scheme.

C. Overlapping scheme

The previous out-of-GPU memory scheme overcomes the restriction of GPU memory, offering the possibility of handling complex electromagnetics (EM) problems. However, the data transfer between CPU and GPU is a time consuming process. Fortunately, the asynchronous technique and CUDA streams can be used to overlap it with the calculation.

Each MPI process opens several CUDA streams [21] on the GPU context (see Fig. 4). The CUDA stream are similar to a CPU pipeline operation queue. Then, matrices B_2 and C_2 are split into smaller matrices according to Fig. 3 (c). These smaller matrices will be transferred to the GPU memory thought different CUDA streams using the CUDA asynchronous data transfer function. After the GPU calculation is completed, the results will be transferred back to RAM in the same way. Note that this transfer process must be executed when the number of CUDA streams is less than the number of smaller matrices.

This overlapping scheme consists of three different operations: data transfer from CPU to GPU, GPU calculation and data transfer from GPU to CPU, these operations are performed by different hardware units. To control the time sequence, the operations in the same CUDA stream must be performed once at a time. However, different operation in different CUDA streams can be done in parallel. Note that the same operation cannot be executed at the same time in two different CUDA streams since they are performed by the same hardware unit.

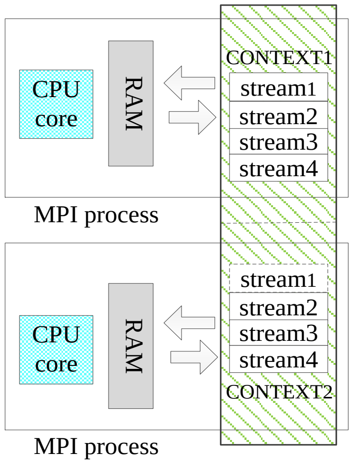


Fig. 4. MPI process opens CUDA streams on GPU context.

For example, in Fig. 5, we have four CUDA streams and the three operation marked with different colors. When one of the streams is involved in data transferring, another can be used for calculations at the same time. The data transfer and calculation of different CUDA streams can be executed in parallel, so the communication time is hidden. Moreover, the only work that CPU does is to start the GPU kernel function. Then CPU will do its own work without waiting for the ending of the GPU computing.

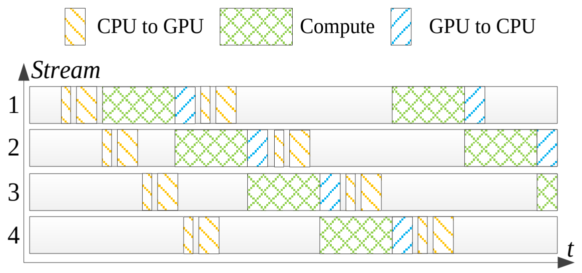


Fig. 5. Overlapping scheme of communication and computation on GPU context.

IV. NUMERICAL RESULTS

In order to demonstrate the correctness and the parallel performance of the proposed hybrid parallel CPU/GPU technique different benchmarks are run. The first test consists of the scattering analysis of PEC sphere used to check the correctness of the implementation.

The second test consists of the analysis of a perfectly conducting cylinder. This test is used to check the parallel performance of the method. Finally, some analysis of a real airplanes and antennas are presented to demonstrate that the method can solve real electromagnetic challenging problems.

The computational platform used for these benchmarks is a high performance GPU cluster with seven computing nodes. Each computing node has two Intel Xeon two 12-core Intel Xeon E5-2692v2 2.2 GHz EM64T processors (12×256 KB L2 Cache and 30 MB L3 Cache), one NVIDIA Tesla K20c GPU card (4.6 GB memory of available) and 64 GB RAM. The nodes are connected with Infiniband switches. The code is developed using the FORTRAN/C/C++ hybrid languages based on MPI.

A. Correctness of the implementation

To validate the accuracy and efficiency of the proposed hybrid parallel CPU/GPU technique the analysis of a PEC sphere with radius of 10 λ is performed. The excitation is a z-axis polarized plane wave propagating along the x-axis direction. The sphere model (see Fig. 6) is discretized into 3258 bilinear patches given a total number of unknowns of 27,528. The bistatic RCS results are given in Fig. 7. A comparison with the analytic Mie solution is performed showing an excellent agreement.

B. Performance testing

The second test consists of the analysis of a perfectly conducting cylinder that is infinitely long along one direction and is illuminated by a transverse magnetic (TM)-polarized plane wave. This benchmark is used to check the performance of the method under different configurations: (1) a single node with 4 MPI processes (each MPI process opens 6 OpenMP threads) and one GPU and (2) two nodes with 8 MPI processes (each MPI process opens 6 OpenMP threads) and two GPUs.

Figure 8 shows the benchmarking results for double precision complex matrices ranging from 1024×1024 to 56320×56320 elements in size. The K20c GPU with 4.6 GB of memory is limited to about 17000 unknowns. Figure 9 shows the benchmarking results for double precision complex matrices ranging from 1024×1024 to 78848×78848 elements in size. The two K20c GPUs with 2×4.6 GB of memory is limited to about 24000 unknowns.

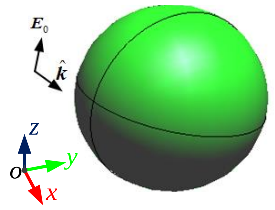


Fig. 6. The model of a PEC sphere.

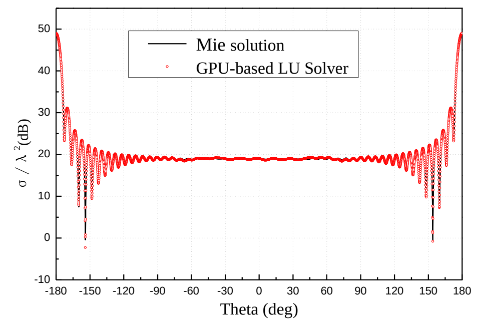


Fig. 7. RCS results in *xoz* plane from the PEC sphere with radius of 10 λ.

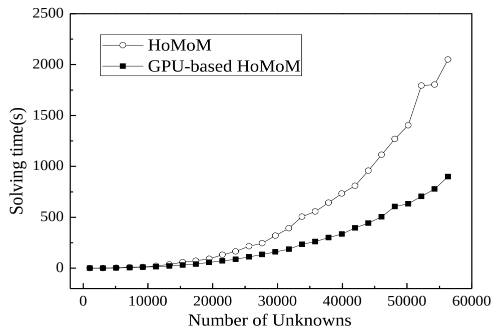


Fig. 8. Performance against matrix size for two versions of LU decomposition using a single node.

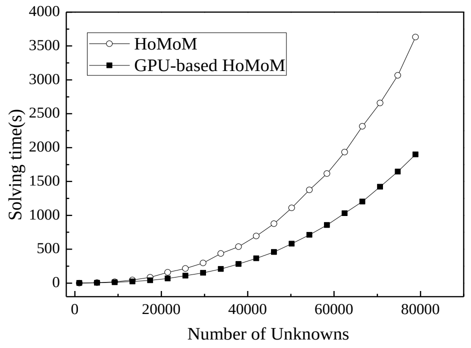


Fig. 9. Performance against matrix size for two versions of LU decomposition using two nodes.

A comparison between the results given by the proposed hybrid parallel CPU/GPU technique and the CPU version only are given in both figures. On a single node, the computing speed of the 24 CPU cores with a single GPU is about 2.3 times than that obtained by using 24 CPU cores alone. On two nodes configuration, the computing speed of the 48 CPU cores with two GPUs is about twice of that from using 48 CPU cores alone. The results show that the proposed technique can save at least 50% of computation time on both distributed and shared memory systems.

C. Performance analysis for metallic structures

This section contains the scattering results of a real airplane. This benchmark demonstrates that the proposed method can solve electromagnetic challenging problem as well. The airplane model is shown in Fig. 10. The airplane is 30.6 m long, 29.0 m wide and 11.8 m high. The bistatic RCS of airplane is simulated at the frequency 440 MHz. The excitation is a *z*-axis polarized plane wave propagating along the negative *x*-axis direction. The airplane is discretized into 16,980 bilinear patches, and the total number of unknowns is 135,501.

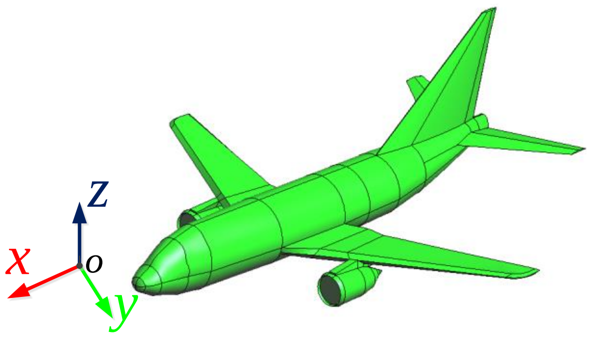


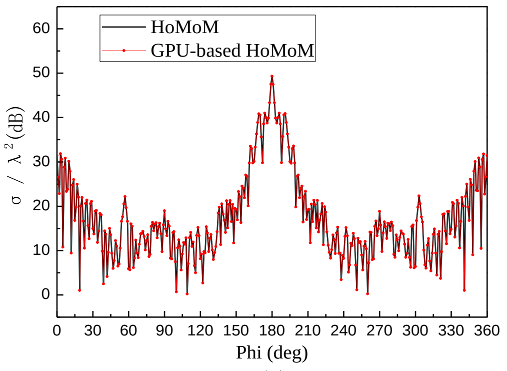
Fig. 10. The model of an airplane.

The two-dimensional (2D) RCS results are shown in Fig. 11. The 2D results computed by the parallel CPU version only are also given for comparison. Note that θ coordinate is measured from *xoy* plane to *z* axis and φ coordinate is measured from +*x* axis to *y* axis in this paper. The computation parameters are listed in Table 1. For this simulation, seven compute node of the described computational platform were used.

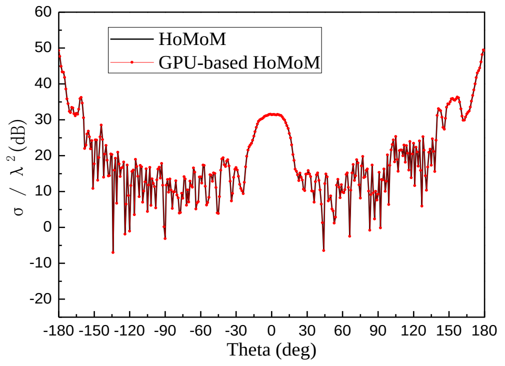
Table 1: Computational parameters for the airplane

Computational Resources	Solving Time (s)	Speedup
24 CPU cores×7	2499.298	1
(24 CPU cores and 1 GPUs) ×7	1089.444	2.294

From the comparisons, one can see that the results of both CPU version only and the proposed hybrid CPU/GPU technique present a very good agreement. The required memory of this simulation is about 274 GB when the memory provided by the GPUs is less than 34 GB. Thus, the proposed algorithm breaks the limitation of the memory of the GPUs as it was described previously. Regarding, the speedup between both codes, the hybrid CPU/GPU code is over 2 times faster, while a speedup of over 380 times is achieved compared to the sequential CPU version only.



(a)



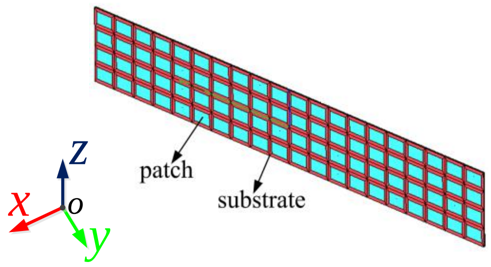
(b)

Fig. 11. 2D RCS of the airplane: (a) *xoy* plane and (b) *xoz* plane.

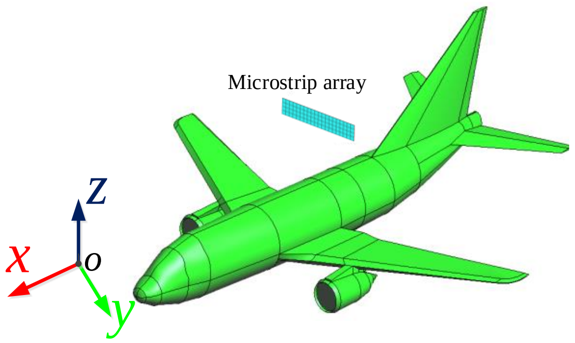
D. Performance analysis for composite metallic and dielectric structures

Finally, the radiation pattern of an airborne array is presented. A microstrip array (20x4) is printed on a substrate ε_r = 4.5 and μ_r = 1.0 and is housed in a 5.27 m by 0.9524 m by 0.018 m cavity in a ground plane, as shown in Fig. 12 (a). The feeding line for each patch has the radius of 1.8 mm. The dimensions of each patch element are 0.2056 m by 0.1548 m, and the gaps between any two neighboring elements are 0.0579 m by 0.0833 m along the length and width directions. The microstrip array is installed 4.0 m above the airplane, as shown in Fig. 12, and the distance between the center of the array and the nose of the airplane is 15.4 m. A -30 dB Taylor amplitude distribution is utilized in the array feed along the *y*-direction and the mainlobe is also directed towards the tail. The operation frequency of the array is 440 MHz. The airborne model is discretized into 21,602 bilinear patches, and the total number of unknowns is 155,494.

The 2D and 3D gain patterns obtained by the proposed method are shown in Fig. 13. The 2D gain patterns computed by the CPU version only are also given for comparison where a very good agreement is clearly seen. In addition, the computation parameters are listed in Table 2.

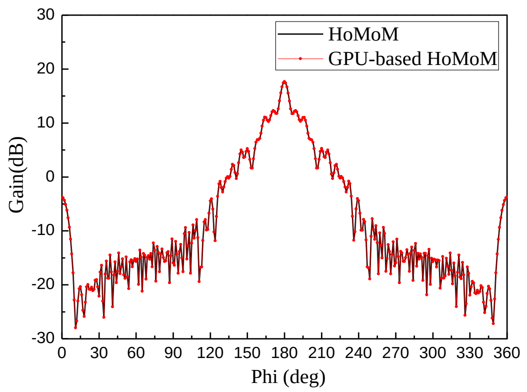


(a)

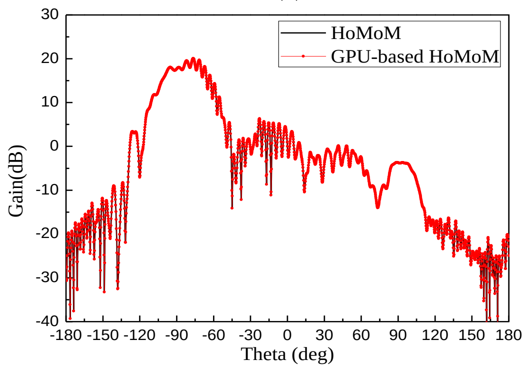


(b)

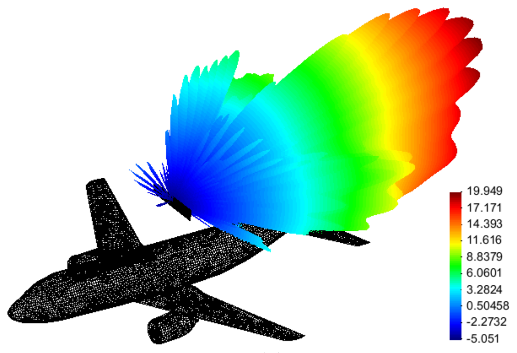
Fig. 12. The airborne array model: (a) the microstrip patch array with 20×4 elements, and (b) the airborne microstrip patch array.



(a)



(b)



(c)

Fig. 13. 2D and 3D gain patterns of the airborne array antenna: (a) *xoy* plane, (b) *xoz* plane, and (c) 3D pattern.

Table 2: Computational parameters for the airborne array

Computational Resources	Solving Time (s)	Speedup
24 CPU cores×7	3749.444	1
(24 CPU cores and 1 GPUs) ×7	1577.735	2.376

In this simulation, we can also see that the limitation of the memory of the GPUs is broken. The required memory of the airborne array (about 360 GB) is quite larger than the available memory of the GPUs (about 34 GB). Moreover, a speedup of over 2 times is achieved compared to the parallel CPU version only, while compared with the sequential HoMoM version, the speedup of the hybrid CPU/GPU technique is about 400 times.

V. CONCLUSION

In this paper, a hybrid parallel CPU/GPU HoMoM method is presented and used to simulate the scattering of an airplane and the radiation pattern of an airborne array. The hybrid parallel CPU/GPU procedure is proved to have a good speedup in the same degree of accuracy compared with the parallel HoMoM using the Intel MKL LU solver. The MPI/OpenMP/CUDA parallel framework base on GPU context technique can support a large scale of parallelism, which can fully exploit the computing power of current distributed CPU/GPU clusters. The out-of-GPU memory scheme overcomes the GPU memory limitation is to utilize both the CPU and GPU memory, which offering a possibility of handling complex electrically large objects.

ACKNOWLEDGMENT

This work was supported in part by the China Postdoctoral Science Foundation funded project under Grant 2017M613068, in part by the National Key Research and Development Program of China under

Grant 2017YFB0202102, in part by the International S&T Cooperation Program under Grant 2016YFE0121600, and in part by the Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the second phase) under Grant No. U1501501.

REFERENCES

[1] Y. Zhang and T. K. Sarkar, *Parallel Solution of Integral Equation Based EM Problems in the Frequency Domain*. Hoboken, NJ: Wiley, 2009.

[2] "NVIDIA GeForce 8800 GPU architecture overview," NVIDIA Corporation, Santa Clara, CA, Tech. Brief TB-02787-001_v0.9, Nov. 2006.

[3] S. E. Krakiwsky, L. E. Turner, and M. M. M. Okoniewski, "Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU)," *in IEEE MTT-S Int. Microwave Symp. Digest*, pp. 1033-1036, 2004.

[4] M. J. Inman and A. Z. Elsherbeni, "Programming video cards for computational electromagnetics applications," *IEEE Antennas Propag. Mag.*, vol. 47, pp. 71-78, 2005.

[5] M. J. Inman, A. Z. Elsherbeni, J. G. Maloney, and B. N. Baker, "Practical implementation of a CPML absorbing boundary for GPU accelerated FDTD technique," *ACES Journal*, vol. 23, no. 1, pp. 16-22, Dec. 2008.

[6] T. Takahashi and T. Hamada, "GPU-accelerated boundary element method for Helmholtz' equation in three dimensions," *International Journal for Numerical Methods in Engineering*, vol. 80, pp. 1295-1321, 2009.

[7] S. Peng and Z. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Trans. Antennas Propag.*, vol. 56, no. 7, pp. 2130-2133, May 2008.

[8] M. J. Inman, A. Z. Elsherbeni, and C. J. Reddy, "CUDA based LU decomposition solvers for CEM applications," *ACES Journal*, vol. 25, no. 4, pp. 339-347, Dec. 2010.

[9] E. Lezar and D. Davidson, "GPU acceleration of method of moments matrix assembly using Rao-Wilton-Glisson basis functions," *in Proc. ICEIE*, Kyoto, Japan, pp. V1-56-V1-60, 2010.

[10] E. Lezar and D. Davidson, "GPU-accelerated method of moments by example: Monostatic scattering," *IEEE Antennas Propag. Mag.*, vol. 52, no. 6, pp. 120-135, Dec. 2010.

[11] E. Lezar and D. B. Davidson, 'GPU-based LU decomposition for large method of moments problems," *Electronics Letters*, vol. 46, no. 17, pp. 1194-1196, 2010.

[12] S. M. Rao, D. R. Wilton, and A. W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. Antennas Propag.*, vol. AP-30, no. 3, pp. 409-418, May 1982.

[13] Innovative Computing Laboratory, University Tennessee, Knoxville, "MAGMA: Matrix Algebra on GPU and Multicore Architectures,"2009. [Online]. Available: http://icl.cs.utk.edu/magma/index.html

[14] E. DAzevedo and JC Hill, 'Parallel LU factorization on GPU cluster," *Procedia Computer Science*, 9, pp. 67-75, 2012.

[15] P. Du, S. Tomov, and J. Dongarra, "Providing GPU Capability to LU and QR within the ScaLAPACK Framework," 2012. [Online] Available: http://www.netlib.org/lapack/lawnspdf/ lawn272.pdf

[16] T. Topa, "Efficient out-of-GPU memory strategies for solving matrix equation generated by method of moments," *Electronics Letters*, vol. 51, no. 19, pp. 1542-1544, 2015.

[17] X. Mu, H.-X. Zhou, K. Chen, and W. Hong, "Higher order method of moments with a parallel out-of-core LU solver on GPU/CPU platform," *IEEE Trans. Antennas Propag.*, vol. 62, no. 11, pp. 5634-5646, 2014.

[18] P. Ylä-Oijala, M. Taskinen, and S. Järvenpää, "Analysis of surface integral equations in electromagnetic scattering and radiation problems," *Engineering Analysis with Boundary Elements*, vol. 32, no. 3, pp. 196-209, 2008.

[19] R. F. Harrington, "Boundary integral formulations for homogenous material bodies," *Journal of Electromagnetic Waves and Applications*, vol. 3, no. 1, pp. 1-15, 1989.

[20] John L. Volakis and Kubilay Sertel, *Integral Equation Methods for Electromagnetics*. Raleigh, NC: SciTech Pub., 2012.

[21] NVIDIA Corporation, 'CUDA API REFERENCE MANUAL Version 5.0', Oct. 2012.