Progress In Electromagnetics Research, Vol. 137, 87–99, 2013

PARALLEL SHOOTING AND BOUNCING RAY METHOD ON GPU CLUSTERS FOR ANALYSIS OF ELECTROMAGNETIC SCATTERING

Pengcheng Gao, Yubo Tao*, and Hai Lin

State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou 310058, P. R. China

Abstract—This paper proposes an efficient parallel shooting and bouncing ray (SBR) method on the graphics processing unit (GPU) cluster for solving the electromagnetic scattering problems. At each incident direction, the parallel SBR method partitions the virtual aperture into sub-apertures, and distributes the computational process of each sub-aperture over GPU nodes. As ray tubes in the virtual aperture do not have the same computational time, the parallel efficiency highly depends on how to partition the virtual aperture. This paper addresses this issue by a dynamic partitioning scheme according to the computational time at the previous angle, which can achieve excellent load balance. Numerical examples are presented to demonstrate the accuracy, high parallel efficiency, good scalability and versatility of the proposed method.

1. INTRODUCTION

The shooting and bouncing ray (SBR) method [1] is widely used for analysis of electromagnetic scattering by large complex objects. Although the SBR is more effective than the numerical approaches (e.g., MOM, FEM), it still is time-consuming for electrically large and complex targets due to that the density of ray tubes on the virtual aperture should be greater than about ten rays per wavelength to ensure the convergence of results, especially for applications in the terahertz (THz) [2, 3]. As a result, various techniques have been proposed to reduce the computational time of the SBR. The octree and kdtree [4, 5] were utilized to accelerate the ray tube tracing of the SBR, and the latter has been well known as the best general-purpose

Received 15 January 2013, Accepted 5 February 2013, Scheduled 13 February 2013

* Corresponding author: Yubo Tao (taoyubo@cad.zju.edu.cn).

acceleration structure for ray tracing of static scenes in computer graphics [6]. The multiresolution grid algorithm [7] was introduced to reduce the total number of ray tubes. Recently, the graphics processing unit (GPU) has become a highly-parallel computational resource and provides enormous performance benefits over CPUs for scientific computing. The SBR worked well with the GPU thanks to the independence of ray tubes [8, 9].

Many real applications involve solving large-scale electromagnetic problems, but it is very difficult or even impossible to solve these problems on a single computer. Parallel computing plays an increasingly important role in today's computational electromagnetics. Previous parallel algorithms (e.g., parallel FDTD [10, 11], parallel direct solver [12] and parallel MLFMM [13, 14]) were mainly implemented on CPU clusters. Due to the high performance/cost ratio and the fast performance growth of GPUs, GPU clusters are becoming more and more popular. The first and the eighth of the top ten systems are powered by NVIDIA GPUs on the Top500 fastest supercomputer list released in November 2012. In fact, the GPU cluster has been applied in the general-purpose computation and electromagnetic simulation applications [15, 16]. As researchers have accelerated a wide range of approaches on the GPU [17–20], the need arises for using the GPU cluster to solve larger scattering problems faster. The question is how to redesign the serial algorithm and extend the scalability of the parallel algorithm to best exploit the potential of these high-performance GPU clusters. Load balance is the key factor to accomplish these goals, especially for heterogeneous GPU clusters equipped with different kinds of GPUs.

This paper proposes a parallel SBR method on the GPU cluster. The virtual aperture is divided into sub-apertures at each incident direction, and the process corresponding to each sub-aperture is distributed to different GPU nodes. Load balance is solved by dynamically partitioning the aperture based on the computational time at the previous angle. In this way, the proposed method ensures the accuracy and achieves high efficiency, good scalability and versatility.

2. PARALLEL SBR SCHEME

2.1. GPU-based SBR

As ray tubes are independent of each other, the SBR can be easily implemented on the GPU with a high degree of parallelism. The procedure of the GPU-based SBR involves three steps: ray tube tracing, electromagnetic computing and parallel reduction [8]. The incident plane wave is modeled as a dense grid of ray tubes, which are

shot toward the target. Each corner ray of ray tubes is recursively traced to obtain the intersection points. The kd-tree augmented with ropes and the stackless kd-tree traversal algorithm [21] are utilized to accelerate ray tracing on the GPU. In the electromagnetic computing step, the first part is to check the validity of the ray tubes. Then, the central rays of ray tubes are also recursively traced like the corner rays, and the fields of them are obtained by the theory of geometrical optics (GO). The scattered filed is calculated by the physical optics (PO) integral, and the formula at an observation point $(r,\ \theta,\ \phi)$ is given:

$$\mathbf{E}(r,\theta,\phi) \approx \frac{e^{-jkr}}{r}(\hat{\theta}E_\theta + \hat{\phi}E_\phi). \tag{1}$$

The $(E_\theta,\ E_\phi)$ can be represented as the exit field (E, H) of the four-sided polygon S:

$$\begin{bmatrix} E_\theta \\ E_\phi \end{bmatrix} = \left(\frac{jk}{2\pi}\right) \iint_S e^{j\mathbf{k}\cdot\mathbf{r}'} \left\{ \begin{bmatrix} -\hat{\phi} \\ \hat{\theta} \end{bmatrix} \times \mathbf{E}(r') f_e + Z_0 \begin{bmatrix} \hat{\theta} \\ \hat{\phi} \end{bmatrix} \times \mathbf{H}(r') f_h \right\} \cdot \hat{\mathbf{n}} dx' dy'. \tag{2}$$

Equation (2) contains three forms with different values of the coefficients f_e and f_h. As indicated in [22], the EH formula provides the best approximation for the PO induced currents.

Finally, when the scattered fields of ray tubes are obtained, the parallel reduction is applied to get the scattered field of the target by summing up these scattered fields.

2.2. Parallel Strategy and Load Balance

In the parallel algorithm without explicit load balancing scheme, the whole task is divided into many sub-tasks, which are scheduled to different computing nodes by the master node during runtime. The size of the sub-task should be small to achieve good load balancing between the computing nodes. However, the number of rays corresponding to the small sub-task may be not sufficient to hide the memory latency, and this will result in decrease in GPU performance. Thus, there is a conflict between them. Owing to that the master node needs to constantly detect the status of the computing nodes, frequent communication between them is required. Additionally, some special cases of the communication should be taken into account (e.g., in some cases synchronization among nodes is necessary to avoid competition). The development of the communication module requires low level network API to meet all the requirements mentioned above and achieve high performance. This means the development will become

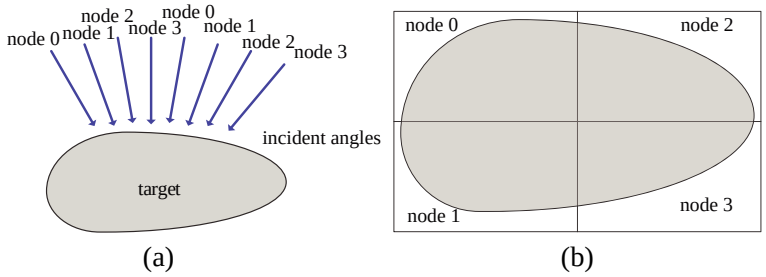

Figure 1. Two parallel schemes. (a) The angle-cyclic method. (b) The virtual aperture partitioning method (uniform partitioning).

a complicated work. In order to avoid the disadvantages of the above method, the distribution for the sub-tasks is determined in advance to minimize the communication between nodes, and an explicit algorithm is designed to adjust the size of the sub-task during runtime to achieve excellent load balancing. The message passing interface (MPI) is utilized to further simplify the implementation.

There are two parallel strategies to the GPU-based SBR. As shown in Figure 1(a), the first one is to distribute the computational processes of consequent incident angles to different GPUs (i.e., an angle-cyclic distribution procedure). The other is to divide the virtual aperture into sub-apertures according to the number of GPUs at each incident angle, as illustrated in Figure 1(b). The first strategy is based on the observation that the computational loads of neighboring angles are almost the same. However, the number of the angles may be not an integral multiple of the number of GPUs. In order to avoid making some GPUs idle while the others are busy, the virtual aperture partitioning scheme should be applied for the remaining angles after even angle distribution. Additionally, with the rapid development of the hardware, there are architecturally distinct GPUs in the cluster, and it results in performance differences among GPUs. In this situation, it can not achieve good load balance by equally distributing angles to GPUs. Thus, the virtual aperture partitioning strategy is more universal than the angle-cyclic strategy, and is used in this paper.

A straightforward implementation of the virtual aperture partitioning strategy is to divide the virtual aperture into uniform sub-apertures with the same number of ray tubes. The full virtual aperture is first split into two sub-apertures with an axis-perpendicular line on the axis with a longer extent. If we assume that each ray tube has the same workload, the split position is $s = (\lfloor N/2 \rfloor / N) \times L$, where N is the number of GPUs that work on this virtual aperture, and

L is the length of the split edge of the virtual aperture. In order to match the workload, the left sub-aperture will be computed over $\lfloor N/2 \rfloor$ GPUs, and the right one will be assigned to $\lceil N/2 \rceil$ GPUs. The two sub-apertures are then recursively partitioned with the updated virtual aperture and the number of assigned GPUs until $N = 1$. Figure 2(a) shows the procedure of the virtual aperture partitioning, and this procedure also constructs a binary tree. The interior node is represented by the splitting line. Its corresponding sub-aperture is split into two-apertures which represent its two children nodes. The leaf nodes represent the sub-apertures distributed to each GPU.

However, the virtual aperture is not fully occupied by the projection of the target, and the projection region is also not evenly distributed to each sub-aperture. The number of intersected ray tubes, which represents the computational load of the SBR, is proportional to the projected area of the target. As illustrated in Figure 1(b), large differences of the projected areas among sub-apertures generated by this uniform partitioning would lead to load imbalance among GPU nodes of the cluster. As the projections at neighboring angles are highly similar, and the computational time of corresponding ray tubes is also almost the same, we can use the execution time of each sub-aperture at the previous angle to dynamically adjust the partitioning of the virtual aperture [23].

The procedure of the dynamic partitioning scheme is described in detail in the following paragraphs. At the previous angle, the computational time of each sub-aperture corresponding to the leaf node is recorded. At the current angle, we first compute the execution time of the sub-aperture corresponding the interior node at the previous angle. It can be evaluated by traversing the tree from leaf to root and recursively summing its left and right child's computational time, i.e., t_l and t_r. Then, we compute the average execution time of a row or column of ray tubes (based on its parent's split axis) for each tree node, except the tree root. Finally, we recursively adjust the split position from the root node to each interior node. For example, the dynamic adjustment of the split position starts from the root node s_0 in Figure 2. As we already assign $n_l = \lfloor N/2 \rfloor$ and $n_r = \lceil N/2 \rceil$ GPUs to its left and right children, the computational time corresponding to its left and right children should have the ratio n_l/n_r. Based on the workload estimation from the previous angle, the adjustment is calculated by the following equation:

$$\frac{t_l + \bar{t}_l \cdot n}{t_r - \bar{t}_r \cdot n} = \frac{n_l}{n_r}, \tag{3}$$

where n represents the amount of the workload adjusted between the two nodes, and $\bar{t}_l$ and $\bar{t}_r$ are the average execution times of the left and

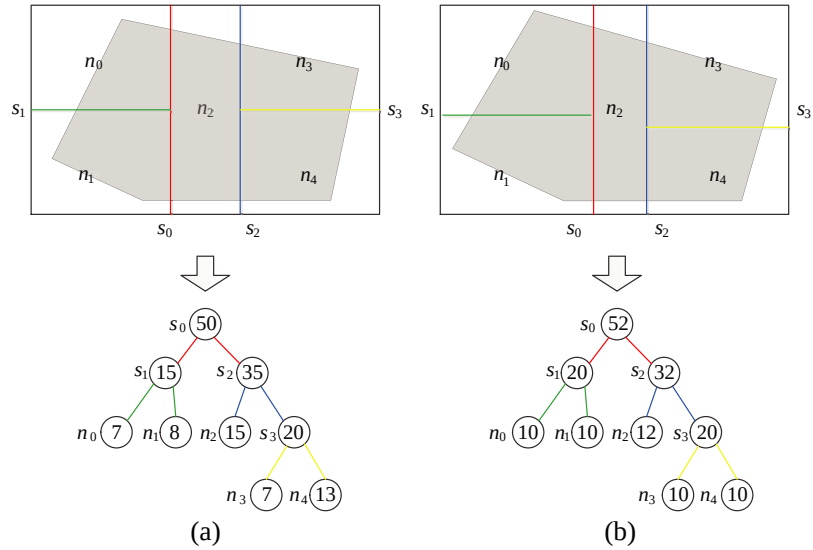

Figure 2. Dynamically adjusting the partitioning of the virtual aperture to achieve good load balance during two subsequent angles. Interior nodes are labeled as their splitting lines and leaf nodes are labeled in their boxes. The numbers represent the computational time of each node.

right child nodes. According to Equation (3), the good load balance can be achieved by ensuring the execution time is proportional to the computational resources (the number of GPUs). Assuming the split position is s_p at the previous angle, the split position s_c of the current angle is expressed as:

$$s_c = s_p + \frac{t_r \cdot n_l - t_l \cdot n_r}{\bar{t}_l \cdot n_r + \bar{t}_r \cdot n_l}. \tag{4}$$

This is the split position of the root node, and the adjustment process is recursively applied to its left and right children until the leaf node. For example, when considering the left child node s_1, because the computational time of n_1 is longer than n_0, the line is moved towards n_1, as shown in green on Figure 2. The two nodes n_0 and n_1 both are the leaf nodes and need no partitioning. The right child node s_2 is processed in the same way, and the splitting line (the blue line) between n_2 and s_3 is moved towards n_2 to reduce the workload for n_2 and increase the workload for s_3. The split position (the yellow line) is also adjusted since the workload of n_4 is more than n_3. This partitioning achieves better load balance compared to the previous angle.

Note that Equation (3) is based on the assumption that the computational time of any rows or columns of a sub-aperture is the same. However, the projection of the target usually just occupies the central area of the virtual aperture, and the border area is empty. The assumption may be not valid in all situations, and the proposed adjustment method may result in over-adjustment. We apply two approaches to avoid the problem. The ideal computational time of each sub-aperture in the interior node is $(t_l + t_r)/2$, and the difference between t_l (or t_r) and the ideal time is $|t_l - t_r|/2$. Thus, the first approach is to adjust the size of the sub-aperture only when the ratio of the above two values, $|t_l - t_r|/(t_l + t_r)$, is higher than the threshold value defined by the user. The other is that n is multiplied by the coefficient in [0, 1] to gradually approach the balance without over-adjustment.

In summary, the procedure of the proposed parallel SBR method is as follows: at each incident direction, the virtual aperture is divided into sub-apertures, and then the process of each sub-aperture is distributed to different GPU nodes. After finishing the calculation of the scattered field using the GPU-based SBR (Section 2.1), the execution time of each sub-aperture is recorded. The computational time of all sub-apertures is broadcast to each GPU node by the MPI, and each node adjusts the split position with this computational time. For the first angle, the aperture is partitioned uniformly, and the aperture is dynamically adjusted based on the computational time at the previous angle for the following angles. After finishing all angles, the scattered fields are gathered to one node.

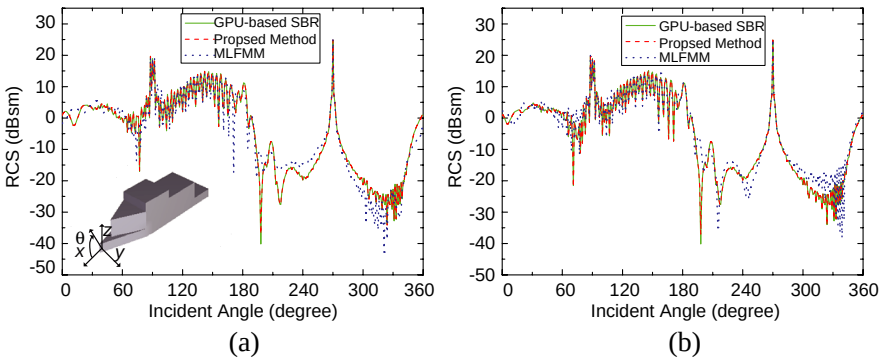

Figure 3. The comparison of the parallel SBR method result, the GPU-based SBR result and the MLFMM result for the ship at 10 GHz. (a) VV-polarization result. (b) HH-polarization result.

3. NUMERICAL RESULTS

To validate the accuracy, efficiency, scalability and versatility of the proposed parallel SBR method, several numerical examples are tested. Experiments presented in this paper were performed on a GPU cluster composed of six computing nodes. Each node is a 1U rack-mount server with 4 GPUs dedicated to computation, and has 24 GB of memory and two Intel Xeon X5650 hexa-core processors with 2.67 GHz clock rates. The GPUs in five nodes are NVIDIA Tesla C2050, and those in last node are NVIDIA GeForce GTX 580. The proposed method is implemented by the CUDA and the MPI.

The ship illustrated in Figure 3(a) is a typical benchmark target for verifying the accuracy of the SBR [4, 7]. The monostatic RCS of the ship at 10 GHz is calculated with the GPU-based SBR, the proposed algorithm, and the MLFMM, respectively. The geometry size of the ship is 0.9 m × 0.2 m × 0.2 m, and the incident parameters are θ from 0° to 360° on the $\phi = 0°$. As can be seen clearly from Figure 3, there is a good agreement between the GPU-based SBR result and the proposed parallel SBR result, and they both agree well with the MLFMM result. The only considerable disagreements between them partly due to the lack of edge-diffraction effect in the SBR results [8]. The monostatic RCS results of another ship are also shown in Figure 4. The geometrical model of the ship is illustrated in Figure 5(a), and the incident direction is rotated around the Y axis from 0° to 90° with the interval of 1°. A good agreement is observed between the two results.

Several different types of targets were tested on five computing

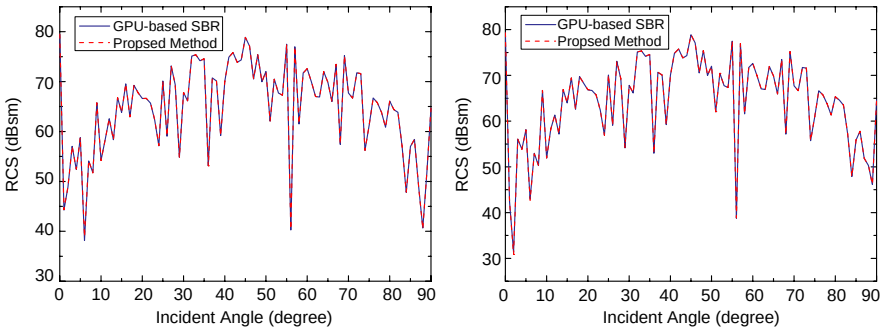

Figure 4. The comparison of the parallel SBR method result and the GPU-based SBR result for the ship at 30 GHz. (a) VV-polarization result. (b) HH-polarization result.

nodes equipped with 20 Tesla C2050 to evaluate the efficiency and scalability of the proposed parallel SBR. As shown in Figure 5, there are a ship, a satellite, an airplane and a radar. The frequency of computation, geometry size and triangle number of the four targets are listed in Table 1. As can be seen from Table 1, the four targets vary in the geometry shape and triangle numbers.

The monostatic RCS of the four targets were calculated, and the incident directions for them are also illustrated in Figure 5. The incident directions for the ship and radar are rotated around the Y axis form 0° to 90° with the interval of 1°, while the others are from 0° to 360° in 361 equal-spaced incident directions. At most fifth-order reflection was considered for complex structures of the four targets. As analyzed in Section 2.2, the dynamic partitioning is performed only when the ratio is higher than 0.05, and n in Equation (3) is multiplied by the coefficient 0.5.

The computational time of the four targets is shown in Table 2 using the GPU-based SBR, the parallel SBR with uniform partitioning, and the parallel SBR with dynamic partitioning. Table 2 indicates that dynamically adjusting the partitioning of the virtual aperture achieves better load balance compared with the uniform partitioning. The parallel efficiency of the airplane is lower compared with the other targets. The reason is that the front part of the fuselage is slender, and the projection of this part is concentrated in the central

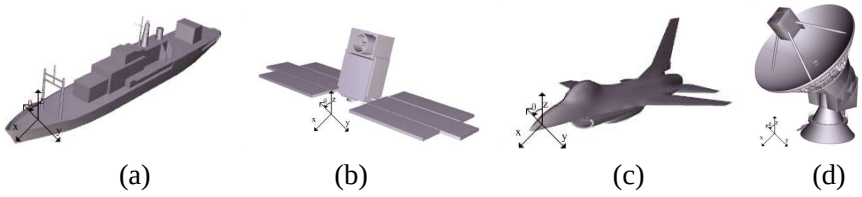

Figure 5. Four test targets: (a) ship, (b) satellite, (c) airplane, (d) radar.

Table 1. The frequency of computation, geometry size and triangle number of the four targets.

Target	Size (m)	Triangle Number	Frequency (GHz)
Ship	$43.71 \times 5.89 \times 9.144$	951	30
Satellite	$35.6 \times 11.64 \times 10.45$	976	15
Airplane	$11.76 \times 7.4 \times 3.67$	13050	30
Radar	$2.66 \times 2.75 \times 3.4$	113374	50

Table 2. The computational time of the four targets of the GPU-based SBR, the parallel SBR with uniform partitioning, and the parallel SBR with dynamic partitioning (Seconds).

Method	Ship	Satellite	Airplane	Radar
GPU-based SBR	517.1	755.8	413.9	554.3
uniform partitioning	38	60.3	46.9	58.9
dynamic partitioning	28.8	41.5	38.8	38
parallel efficiency (uniform)	75%	62.7%	44%	47%
parallel efficiency (dynamic)	90%	91%	53.3%	73%

area of the virtual aperture. A small movement of the split position in the central area can lead to a big change of computational time. Therefore, the coefficient multiplied by n is reduced from 0.5 to 0.3 to avoid the over-adjustment for the airplane. The parallel efficiency is correspondingly increased to 75%. Although the coefficient is a adjustable parameter, the value of 0.5 is optimal for the majority of targets in our experiments. For the targets whose scales of each part are almost equal and the projections occupy the majority of the area of the virtual aperture (e.g., the ship and satellite), the parallel efficiency of the proposed method is very high. The efficiency can reach up to more than 70% by easily adjusting the coefficient even for the targets, there are large differences among the dimensions of their each part (e.g., the fuselage and the wings of the airplane).

Figure 6 shows the speed-up of the parallel SBR method for the four targets. A linear speed-up represents a good scalability of the parallel algorithm, and the ideal situation is that the slope of the linear speed-up is one. In Figure 6, the linear growth of speed-up is observed with the increasing number of the GPUs, and the slope is close to one. This shows good scalability as the computational power of GPUs can still be efficiently used when the number of GPUs increases.

The satellite illustrated in Figure 5(b) is also tested on a heterogeneous GPU cluster including two computing nodes equipped with different GPUs (i.e., four Tesla C2050 and four GeForce GTX 580). Figure 7 shows the maximum and minimum computational time among the eight GPUs at each incident angle for the two partitioning methods. Large differences between the maximum and minimum time result in bad load balance among GPUs. As shown in Figure 7, the difference of the computational power of GPUs results in a relatively large disparity of the computational time at the beginning. The large disparity still exists at the following angles for the uniform partitioning scheme. However, the difference is getting smaller due to the dynamic

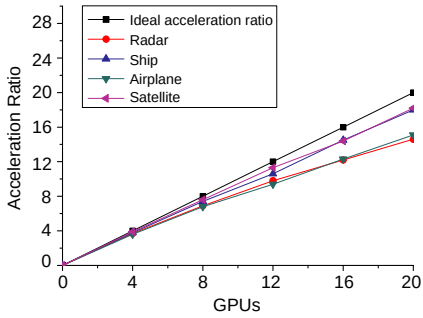

Figure 6. Speed-up of the parallel SBR method for the four targets.

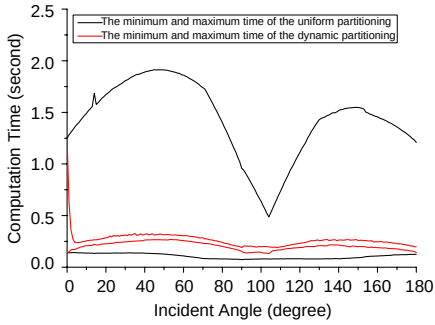

Figure 7. The minimum and maximum computational time for each incident angle.

load adjustment, and the disparities are reduced to a very small range after a few angles. The proposed dynamic partitioning scheme significantly reduces load imbalance. This demonstrates the proposed method is also suitable for the heterogeneous GPU cluster.

4. CONCLUSION

A load-balanced parallel SBR method based on the virtual aperture partitioning scheme is developed on the GPU cluster for analyzing the electromagnetic scattering problems. The dynamic load adjustment strategy is designed to reduce the difference of the computational time among GPUs based on the computational time of each GPU at the previous angle. The numerical results show the accuracy, good parallel efficiency and scalability of the proposed method, and demonstrate that it can also work well on the heterogeneous GPU cluster.

ACKNOWLEDGMENT

The authors would like to thank Prof. T.-J. Cui from South East University for providing the MLFMM method used in this paper. This work was partially supported by NFS of China (No. 61171035) and the Fundamental Research Funds for the Central Universities.

REFERENCES

1. Ling, H., R. C. Chou, and S. W. Lee, "Shooting and bouncing rays: Calculating the RCS of an arbitrarily shaped cavity," *IEEE*

Trans. Antennas Propag., Vol. 37, No. 2, 194–205, 1989.

2. Heh, D. Y. and E. L. Tan, and H. Lin, "Modeling the interaction of terahertz pulse with healthy skin and basal cell carcinoma using the unconditionally stable fundamental adi-FDTD method," *Progress In Electromagnetics Research B*, Vol. 37, 365–386, 2012.
3. Nam, K. M., L. M. Zurk, and S. Schecklman, "Modeling terahertz diffuse scattering from granular media using radiative transfer theory," *Progress In Electromagnetics Research B*, Vol. 38, 205–223, 2012.
4. Jin, K. S., T. I. Suh, S. H. Suk, B. C. Kim, and H. T. Kim, "Fast ray tracing using a space-division algorithm for RCS prediction," *Journal of Electromagnetic Waves and Applications*, Vol. 20, No. 1, 119–126, 2006.
5. Tao, Y. B., H. Lin, and H. J. Bao, "KD-tree based fast ray tracing for RCS prediction," *Progress In Electromagnetics Research*, Vol. 81, 329–341, 2008.
6. Havran, V., "Heuristic ray shooting algorithms," Ph.D. Dissertation, Univ. Czech Technical, Prague, 2000.
7. Suk, S. H., T. I. Seo, H. S. Park, and H. T. Kim, "Multiresolution grid algorithm in the SBR and its application to the RCS calculation," *Microw. Opt. Technol. Lett.*, Vol. 29, No. 6, 394–397, 2001.
8. Tao, Y. B., H. Lin, and H. J. Bao, "GPU-based shooting and bouncing ray method for fast RCS prediction," *IEEE Trans. Antennas Propag.*, Vol. 58, No. 2, 494–502, 2010.
9. Gao, P. C., Y. B. Tao, and H. Lin, "Fast RCS prediction using multiresolution shooting and bouncing ray method on the GPU," *Progress In Electromagnetics Research*, Vol. 107, 187–202, 2010.
10. Vaccari, A., A. Cala'Lesina, L. Cristoforetti, and R. Pontalti, "Parallel implementation of a 3D subgridding FDTD algorithm for large simulations," *Progress In Electromagnetics Research*, Vol. 120, 263–292, 2011.
11. Guo, X.-M., Q.-X. Guo, W. Zhao, and W. Yu, "Parallel FDTD simulation using NUMA acceleration technique," *Progress In Electromagnetics Research Letters*, Vol. 28, 1–8, 2012.
12. Garcia-Donoro, D., I. Martinez-Fernandez, L. E. Garcia-Castillo, Y. Zhang, and T. K. Sarkar, "RCS computation using a parallel in-core and out-of-core direct solver," *Progress In Electromagnetics Research*, Vol. 118, 505–525, 2011.
13. Pan, X.-M., W.-C. Pi, and X.-Q. Sheng, "On openMP parallelization of the multilevel fast multipole algorithm," *Progress*

In Electromagnetics Research, Vol. 112, 199–213, 2011.

14. Ergul, O., "Parallel implementation of MLFMA for homogeneous objects with various material properties," *Progress In Electromagnetics Research*, Vol. 121, 505–520, 2011
15. Fan, Z., F. Qiu, and A. Kaufman, "Zippy: A framework for computation and visualization on a GPU cluster," *Computer Graphics Forum*, Vol. 27, No. 2, 341–350, 2008.
16. Gödel, N., N. Nunn, T. Warburton, and M. Clemens, "Scalability of high-order discontinuous Galerkin FEM computations for solving electromagnetic wave propagation problems on GPU Clusters," *IEEE Trans. Magn.*, Vol. 46, No. 8, 3469–3472, 2010.
17. Lee, K. H., I. Ahmed, R. S. M. Goh, E. H. Khoo, E. P. Li, and T. G. G. Hung, "Implementation of the FDTD method based on Lorentz-Drude dispersive model on GPU for plasmonics applications," *Progress In Electromagnetics Research*, Vol. 116, 441–456, 2011.
18. Shahmansouri, A and B. Rashidian, "GPU implementation of split-field finite-difference time-domain method for Drude-Lorentz dispersive media," *Progress In Electromagnetics Research*, Vol. 125, 55–77, 2012.
19. Dziekonski, A., P. Sypek, A. Lamecki, and M. Mrozowski, "Finite element matrix generation on a GPU," *Progress In Electromagnetics Research*, Vol. 128, 249–265, 2012.
20. Capozzoli, A., C. Curcio, and A. Liseno, "Fast GPU-based interpolation for SAR backprojection," *Progress In Electromagnetics Research*, Vol. 133, 259–283, 2013.
21. Popov. S, J. Günther, H.-P. Seidel, and P. Slusallek, "Stackless KD-tree traversal for high performance GPU ray tracing," *Computer Graphics Forum*, Vol. 26, No. 3, 415–424, 2007.
22. Baldauf, J., S. W. Lee, L. Lin, S. K. Jeng, S. M. Scarborough, and C. L. Yu, "High frequency scattering from trihedral corner reflectors and other benchmark targets: SBR versus experiments," *IEEE Trans. Antennas Propag.*, Vol. 39, No. 9, 1345–1351, 1991.
23. Marchesin, S., C. Mongenet, and J.-M. Dischler, "Dynamic load balancing for parallel volume rendering," *Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization*, 43–50, 2006.