Journal of Innovative Optical Health Sciences
Vol. 5, No. 2 (2012) 1250004 (9 pages)

DOI: 10.1142/S1793545812500046

GRAPHICS PROCESSING UNIT CLUSTER ACCELERATED MONTE CARLO SIMULATION OF PHOTON TRANSPORT IN MULTI-LAYERED TISSUES

CHAO JIANG[*,†], HENG HE[*,†], PENGCHENG LI[*,†,‡]
and QINGMING LUO[*,†]
[*]*Britton Chance Center for Biomedical Photonics*
Wuhan National Laboratory for Optoelectronics
Huazhong University of Science and Technology
Wuhan 430074, China

[†]*Key Laboratory of Biomedical Photonics of Ministry of Education*
Huazhong University of Science and Technology
Wuhan 430074, China
[‡]*pengchengli@mail.hust.edu.cn*

Accepted 18 December 2011
Published 16 March 2012

We present a graphics processing unit (GPU) cluster-based Monte Carlo simulation of photon transport in multi-layered tissues. The cluster is composed of multiple computing nodes in a local area network where each node is a personal computer equipped with one or several GPU(s) for parallel computing. In this study, the MPI (Message Passing Interface), the OpenMP (Open Multi-Processing) and the CUDA (Compute Unified Device Architecture) technologies are employed to develop the program. It is demonstrated that this designing runs roughly N times faster than that using single GPU when the GPUs within the cluster are of the same type, where N is the total number of the GPUs within the cluster.

Keywords: Photon transport in tissues; Monte Carlo simulation; GPU cluster.

1. Introduction

In the field of biomedical optics, it is of great importance to simulate the photon transport in biological tissues.[1] Among the methods usually used for modeling the photon transporting in random media, Monte Carlo (MC) simulation is generally considered as a gold standard for modeling the photon transport in biological tissues because of its flexibility and rigorousness, especially for the biological tissue with complex structures such as adult head, skin and so on. The MCML program developed by Wang *et al.*[2] has already been widely used for the Monte Carlo simulation of light transport in multi-layered tissues since its release. However, Monte Carlo simulation is quite time-consuming due to the extensive computational burden. Fortunately, the algorithm of Monte Carlo method has the full characteristics of parallel computing so that the simulation can be accelerated by parallel computing. Kirkby and Delpy[3] used a 24-computer

network to simulate the light transport in tissue using Monte Carlo method, and Colasanti *et al.*[4] used a CRAY parallel processor computer with 128 processors. The FPGA (Field Programmable Gate Array) is also used for accelerating the Monte Carlo simulation on success by Lo *et al.*[5]

With the appearance of the General-Purpose computation on Graphics Processing Units (GPGPU), the application of GPU on parallel computing has been attracting extensive interests in recent times. The rapid increase in the performance of GPU, together with improvements in its programmability makes it not only work as graphics rendering device but also a kind of powerful coprocessors for general-purposed parallel computing. Alerstam *et al.*[6] performed a Monte Carlo simulation of time-resolved photon migration in homogenous turbid media with semi-infinite geometry by using a low-cost GPU (NVIDIA Geforce 8800). This program does not generate any spatial resolved result. They also developed a method of Monte Carlo simulation of light transport in multi-layered tissues based on GPU called CUDAMCML,[7] which runs several 10 times to around 100 times for various tissue models faster than the MCML running on CPU with a NVIDIA Geforce 9800 GPU. Lo *et al.*[8] used four high-end GPUs (Nvidia GTX 295 and Nvidia GTX 280) in a single computer for acceleration of a standard code for the Monte Carlo simulation of photon transport for multiple layered turbid media and achieved 1052 folds performance improvement. The authors call this multiple GPU-based MC (Monte Carlo) program GPU-MCML. After that, Alerstam *et al.*[9] further optimized the GPU-MCML for a more advanced Nvidia GPU (the Fermi GPU). Like our program, both the CUDAMCML and GPU-MCML are extensions of standard MCML codes of Wang *et al.*[2] to GPU platform and generate the same spatial resolved result as MCML. Besides, GPU is also used to accelerate some other MC simulations of more complex model, such as arbitrary 3D turbid media[10] and heterogeneous tissue model whose surfaces are constructed by different number of triangle meshes.[11]

To further improve the performance of GPU-based Monte Carlo simulation, we developed a Monte Carlo program for simulation of light transport in multi-layered tissues using the GPU cluster. We call it GPU cluster-based MCML or GCMCML for short. In the GCMCML, distributed computing of Monte Carlo simulation is performed in multiple GPUs equipped in multiple computers located in a local area network (LAN). The designing, performance analysis and result validation are discussed. The source code of GCMCML is released at "http://bmp.hust.edu.cn/GPU_Cluster/GPU_Cluster_MCML.HTM".

2. GPU-Based MCML

Our work is based on CUDA (Compute Unified Device Architecture) GPU programming toolkit developed by NVIDIA. CUDA is a new hardware and software architecture for issuing and managing computations running on the GPU as a data-parallel computing device without the need of mapping them to graphics API (Application Program Interface).[12]

In this study, the part of the program running on the GPU or namely kernel is developed using the CUDA. We programmed the skeleton of the kernel by modifying the conventional MCML code developed by Wang *et al.*[2] In our program, when the kernel starts, thousands of threads are launched. The exact optimized number of threads depends on the resources (number of processors and registers) that a specific GPU can provide. The whole simulation is partitioned into many small parts and each thread is responsible for processing one part (a number of photon packets). The rule of photon transport in the tissue model is of the same as that of the conventional CPU-based MCML.[2] The weights left by the photon packets are recorded in the global memory (video memory). The interior absorption portion is recorded in an array called A_rz and the diffuse reflectance and transmittance are recorded in the other two arrays called Rd_ra and Tt_ra, respectively just as the conventional MCML does. After the whole simulation is completed, the host CPU copies the results from global video memory to the main memory. This is the implementing process of the GPU MCML on each node of the cluster.

An important problem for programming Monte Carlo codes for GPU platform is the random number generation. Each thread should own a different seed for generating random number and the random sequence of each thread should be independent of each other. The access of global memory is much slower than register, so a less memory needed algorithm for making use of register should be selected for considering the performance. After comparing several often used algorithms, we select the same method as CUDAMCML called

Multiply-With-Carry (MWC) for its simplicity and long period (more than 2^{60}).[6,13]

To improve the performance of GPU-based MCML kernel of our program as much as possible, some optimizations were performed by referring to the CUDAMCML developed by Alerstam *et al.*[6] and the GPU-MCML developed by Lo *et al.*[8] These optimizations include the use of share memory for buffering part weights to avoid frequently accessing global memory and some solutions for reducing the divergence of the computational kernel.

3. GPU Cluster-Based MCML

In the previous studies for GPU-based MCML, only one computer is used. Usually the commercial personal computer can only support no more than four GPUs in one PC. This limitation hinders the further improvement of the performance of GPU-based MCML. In our study, the technologies of network distributed computing and general-purposed GPU are combined for accelerating the Monte Carlo simulation. This method enables a cluster of computers to cooperate for one simulation task. Each computer of the cluster is called a node and equipped with one or more GPUs. This solution can be considered as a two-layered parallel computing architecture to accelerate the Monte Carlo simulation. The GPUs of each node work as the bottom layer of this architecture and all the nodes of the cluster work as the top layer. The implementation detail of the bottom layer is introduced in the second section of this paper. This section emphasizes the implementation details of the top layer and the whole architecture.

Directly using the basic socket interfaces to do communications in network distributed computing is not a highly efficient way. On one hand, this solution greatly increases the order of complexity of programming; on the other hand, it is not convenient for the program transplanting among different operating systems, as the socket interfaces are not the same in different operating systems (such as Microsoft Windows and Linux). In this study, we use the MPI (Message Passing Interface) technology to communicate among the nodes of the cluster in a LAN. The MPI is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation, whose goals are high performance, scalability and portability.[14] At present, MPI becomes the main model used in network distributed computing for high performance parallel computing.

In the study of Lo *et al.*, they use four GPUs in one computer for the computing. Our GCMCML also enables multiple GPUs on each node of the cluster. Generally, the operating of multiple GPUs on one computer must use the multi-thread programming. However, the interfaces for multi-thread programming are different in different operating systems. This brings some inconvenience for the transplanting of the program among different operating systems. The program must prepare multiple sets of multi-thread generating codes for different operating system like GPU-MCML. In order to keep the savory transplanting among different operating systems, we adopt the OpenMP (Open Multi-Processing) for generating multiple threads. The OpenMP is an application programming interface (API) that supports multi-platform shared memory multi-processing programming in C, C++ and Fortran on many architectures, including Unix and Microsoft Windows platforms. It is much easier and more flexible than the Windows and Unix's own multi-thread interfaces for generating multiple threads.

Figure 1 illustrates the details of the implementation of GCMCML. The whole implementation can be described by six steps. First, the main node (PC0) queries the number of GPUs of each node within the cluster and makes a record in a table. This table will be used for deciding how many random seeds are delivered to a certain node by main node.

Second, the main node (PC0) reads the random seeds and optical properties of the media from the input data files. The random seeds will be used to generate the random number on the GPU. The main node also splits and allocates the task for each node of the cluster on average according to the whole number of photon packets and the number of GPUs on the each node.

Third, the main node scatters the random seeds to other nodes. Each node is assigned certain sets of different random seeds by the main node according to the table built in the step 1. The number of the GPUs of a node decides that how many sets of random seeds it receives. Besides, the main node also broadcasts the media parameters to all other nodes.

Fourth, the host CPU of each node decides how many host threads should be created according to the number of GPUs of this node by using the OpenMP. Each host thread is responsible for

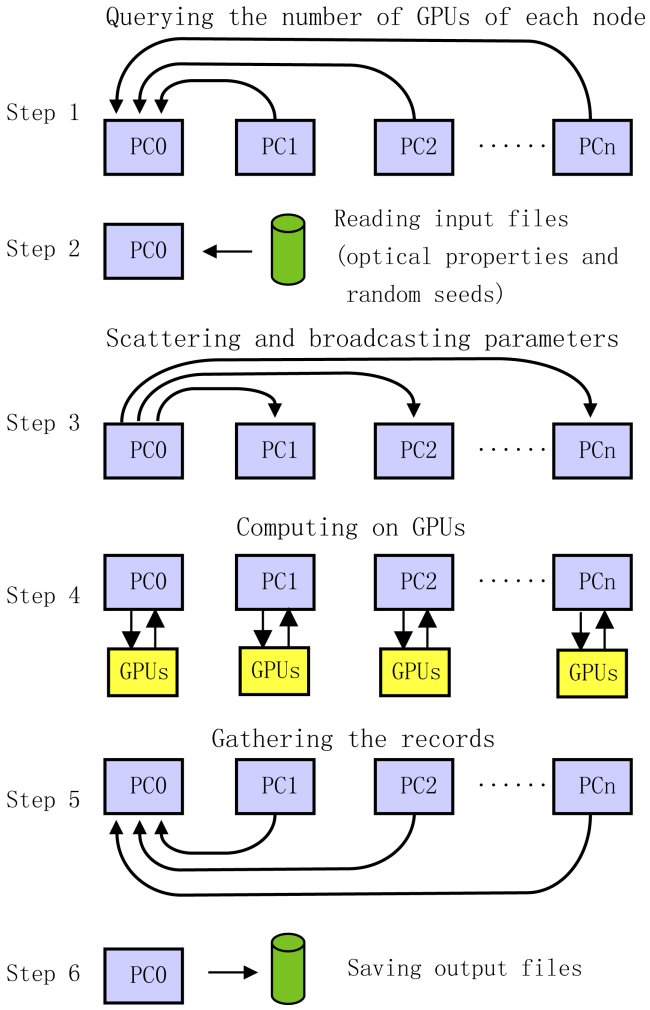


Fig. 1. The implementation skeleton of the GCMCML. Main node (PC0) is responsible for delivering the random seeds and optical properties of the media from the input files to the other nodes, gathering the simulation results of each node into a whole and storing it in the output files. All the nodes including the main node process one part of the whole simulation by employing GPU. The arrow stands for the direction in which the data flows. The GPUs of this cluster are assumed to be of the same type.

operating a GPU. The host threads of each node transfer the respective set of random seeds and the media parameters to the global video memory of the respective GPUs. Then the GPU is started by the respective host thread to conduct the Monte Carlo simulation. After each GPU completes its simulation, the host thread of the node read back the results from the graphic hardware's global memory. If more than one GPU is employed for computing in a node, the main process of the host will also need to sum all the results of the host threads.

Fifth, all the records of the nodes are gathered to the main node. Finally, the main node stores the results into the output data files. The results involve the light distribution absorbed by the media, the diffuse reflectance and the transmittance.

In our program, the main process of the host converts the 64-bit integer sums into 32-bit float data after the records of all the host threads are summed. If the data is directly sent to the main node and sequentially summed on the main node, a large round-off error will happen when the number of the nodes of the cluster is big. In order to avoid this problem, we adopt another parallel partial summation algorithm for summing these records of the different nodes. All the nodes of the cluster cooperate to complete this summing operation. This algorithm is carried out through a serial of successive reduction operations. Each time, half of the nodes send their records to another half of nodes in a one-to-one mode and each node of the latter half of the nodes sums the two sets of the records. Next time, the number of nodes to anticipate this reduction operation halves and the rest of the nodes do the same operation as the last time. After several times, the total sum of the records can be obtained on the main node. In these reduction operations, the number of nodes may not be even at some times. In this case, the main nodes sum up two nodes' records at a time while the rest of the nodes still perform in a one-to-one mode. By using this parallel partial summation algorithm, the round-off errors can be reduced greatly. Besides, the performance is also improved slightly. Figure 2

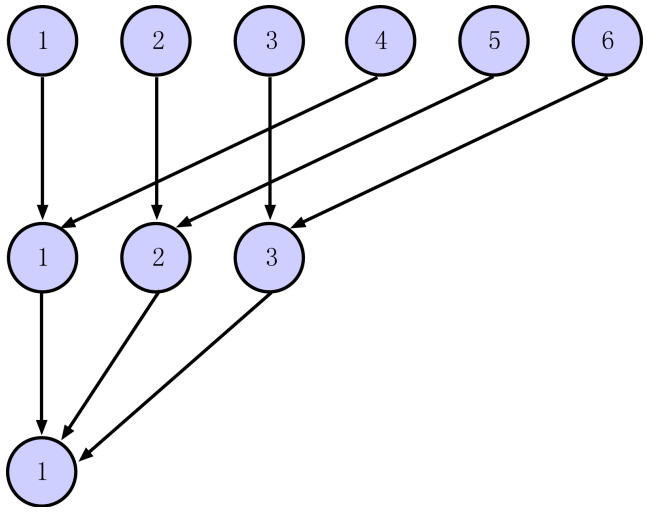


Fig. 2. An illustration of the implementation of parallel reduction method when the total number of the nodes is six. Each circle stands for a node.

gives a simple example of this parallel reduction in the case that the total number of the nodes is six.

4. Results

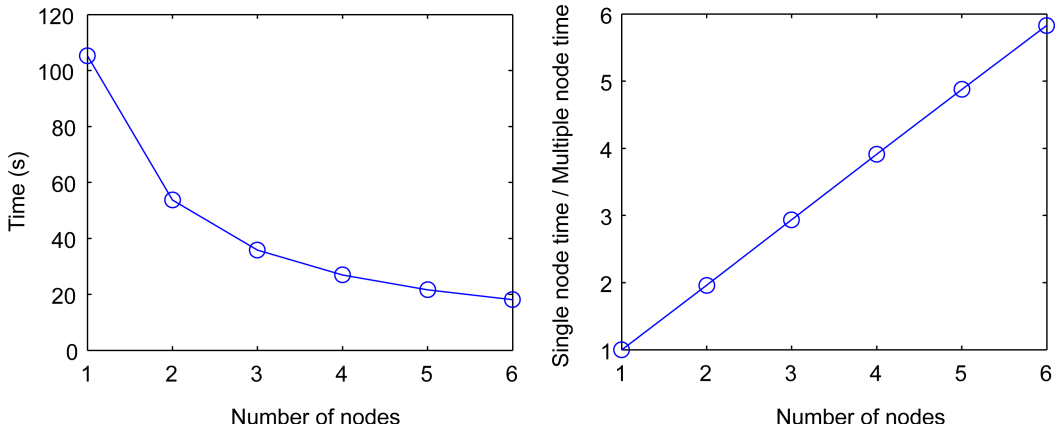
4.1. *Performance*

To evaluate the performance of GCMCML, a 5-layered skin model from Tuchin is used for the simulation.[15] The optical properties are presented in Table 1 involving the absorption coefficients μ_a, the scattering coefficients μ_s, the anisotropy g and the refractive index n. The grid resolution is 0.002 cm for dz and 0.01 cm for dr. The number of the grids is 256 for nr and 256 for nz. The number of the photon packets is 100 million. This skin model is also adopted by Lo *et al.*[5,8] in their evaluation of FPGA-based MCML and GPU-MCML. We conducted the simulation of this skin model on two kinds of GPU cluster. One is composed of six low-end Nvidia GPUs (Geforce 9800GT), and the other is a high-end Nvidia GPU (GTX480). The host CPU adopted in these clusters is Intel E7300 (two-core and made in 45 nm) working at 2.66 GHZ. The program is developed in Microsoft Visual Studio 2005 in Windows XP and the data type is single-precision float point. The results below are obtained from the implementation of this program in Windows XP. Besides, a Makefile for Linux is also provided on our website.

To investigate the relationship between the performance of GCMCML and the number of the nodes, we perform a series of simulations using GCMCML with different number of nodes. Each node is equipped with one single Geforce 9800GT. It is demonstrated that the simulation time consumption decreases with the increasing number of nodes, as shown in Fig. 3(a). The time consumption is roughly inversely proportional to the number of the nodes. Figure 3(b) also presents the speedup fold by different number of the nodes in contrast to single node. It shows that the performance improvement is roughly directly proportional to the number of the nodes. When six nodes are added into the cluster, the simulation can be around

Table 1. Optical properties of the 5-layered skin model at 633 nm.

Layer	n	μ_a (cm^{-1})	μ_s (cm^{-1})	g	Thickness (cm)
Epidermis	1.5	4.3	107	0.79	0.01
Dermis	1.4	2.7	187	0.82	0.02
Dermis with plexus superficialis	1.4	3.3	192	0.82	0.02
Dermis	1.4	2.7	187	0.82	0.09
Dermis plexus profundus	1.4	3.4	194	0.82	0.06



(a) (b)

Fig. 3. The relation between the performance and the total number of the nodes of GCMCML: (a) The time consumption versus the number of nodes within the cluster for simulation of 100 million photon packets in the skin model as Table 1. (b) The ratio between the time consumption using single one node and those using multiple nodes. Each node is equipped with 1 Nvidia Geforce 9800GT.

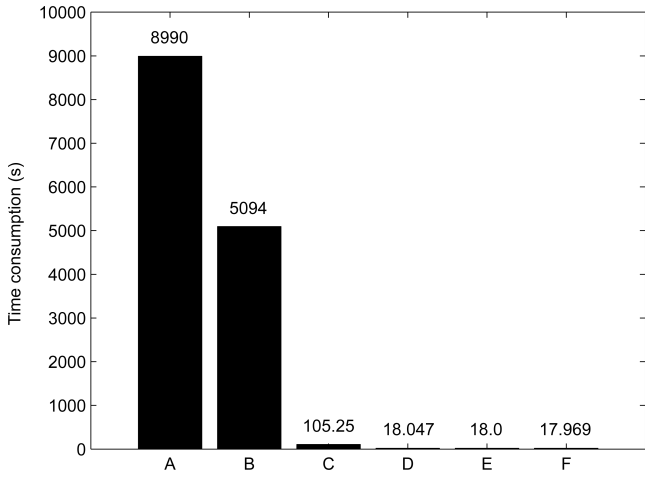


Fig. 4. Time consumption of different combinations of computers and GPUs: A: CPU-based MCML (CPU: Intel E6300, 2-core, 65 nm and 1.86 GHz); B: CPU-based MCML (CPU: Intel E7300, 2-core, 45 nm and 2.66 GHz); C: GCMCML with one GPU; D: GCMCML with six nodes and each node is equipped with one GPU; E: GCMCML with five nodes and one of the five nodes is equipped with two GPUs; F: GCMCML with four nodes and two of the four nodes are equipped with two GPUs. The GPUs in these simulations are Nvidia Geforce 9800GT.

5.83 times faster than using single node. If more nodes are added into the cluster, better performance can be expected.

Our program also enables multiple GPUs on each node where the number of GPUs of different nodes can be different. An example is presented in Fig. 4. Different combinations of computers and GPUs are used to perform the simulation. In this example, the number of computers varies and the total number of the GPUs is kept as six. It is proved that the time consumption is mainly decided by the total number of the GPUs. The number of the nodes nearly has little affection on the performance of the simulation. Besides, the time consumptions of the conventional MCML[2] and GCMCML using single one GPU are also plotted as contrasts. For the conventional MCML tests, two kinds of CPU with different processing ability are adopted. The conventional MCML source codes are recompiled using Microsoft Visual Studio 2005 into Win32 release version executable. The C++ code optimization item of the compilation settings is chose as "Full optimization." The data type is kept as double-precision float point without change, but the random number generator is replaced with MWC for the new recompiled conventional MCML.

A further investigation of time consumptions of different operations is carried out. Table 2 demonstrates that the overheads of the operations of scattering and broadcasting, and gathering are relatively quite small in contrast to that of simulations. The time consumption of the two operations falls with the decrease of the number of the nodes on the total trend. It is interesting that the time consumption of gathering between two neighboring groups is the same when the number of the nodes is more than two. It is caused by the parallel reduction algorithm. Actually, the main node's time consumption on the gathering operation stands for the total time consumption of this operation. From the perspective of the main nodes, the computation burdens of the two neighboring groups are the same.

Another test using a high-end GPU (GTX480) is also carried out. This GPU enables the atomic operations to the share memory. So the optimization of caching the high fluence regions in share memory is allowed. This greatly boosts the total performance of the GCMCML. For the same skin model of Table 1, the time consumption can be reduced to about 9.75 s with only one kind of GPU by GCMCML when 100 million photon packets are simulated. It indicates that the performance gained by 1 GTX480 is roughly equal to that of 11 Geforce 9800GT. If more of these high-end GPUs are added

Table 2. The detailed time consumption of different operations of the simulations.

GPUs	Scattering and broadcasting (s)	Gathering (s)	Simulation (s)	Total time (s)
6	0.031	0.094	17.922	18.047
5	0.016	0.094	21.484	21.594
4	0.016	0.062	26.828	26.906
3	0.016	0.062	35.766	35.844
2	0.016	0.046	53.609	53.671
1	0.0	0.0	105.25	105.25

to the cluster, more powerful performance boost will be gained just like the tests using Geforce 9800GT.

As Alerstam *et al.*[7] and Lo *et al.*[8] point out, the performance improvements are also dependent on the optical properties and the grid resolution. Therefore, the performance improvements are different for different models. In this paper, the performance improvements of our GCMCML for a 5-layered media are about 48 folds on a single Geforce 9800GT, 283 folds on a cluster composed of six Geforce 9800GT and 522 folds on a single GTX 480 in contrast to the conventional MCML on Intel E7300 CPU. We also did a lot of tests using other models with different optical properties and grid resolution. The performance improvements of some models are higher than that of the 5-layered skin model while others are lower.

4.2. *Validation*

As Lo *et al.*[5] refers, MC simulations are non-deterministic. The simulation results of GCMCML were validated against those of the conventional gold standard MCML[2] with the statistical uncertainty. The relative differences of the internal absorption, the diffuse reflectance and the transmittance from the two version MC programs are analyzed. All the data comes from the simulation result of the same 5-layered skin model above with 100 million photon packets.

Figure 5 illustrates the contours distribution of the internal absorption. The contours of the same value between GCMCML and MCML consist well in total and notable difference can hardly be observed from Fig. 5(a). Figures 5(b) and 5(d) are the enlarged pictures of the contours of 0.01 in Figs. 5(a) and 5(c), respectively. Figure 5(b) shows that there is a small shift for the two contours (red dots and blue line). This shift also exists for the comparison of the two simulations using the same CPU-based MCML with different random sequence, as shown in Fig. 5(d). We also found that there are nearly no shift for the other five contours both for the comparisons of GCMCML and MCML and two MCML with different random sequences (here we did not plot). It indicates that this kind of shift becomes notable when the value is becoming very small, such as 0.01. As Lo *et al.*[5] refers, these differences are the statistical uncertainties between runs due to the different random

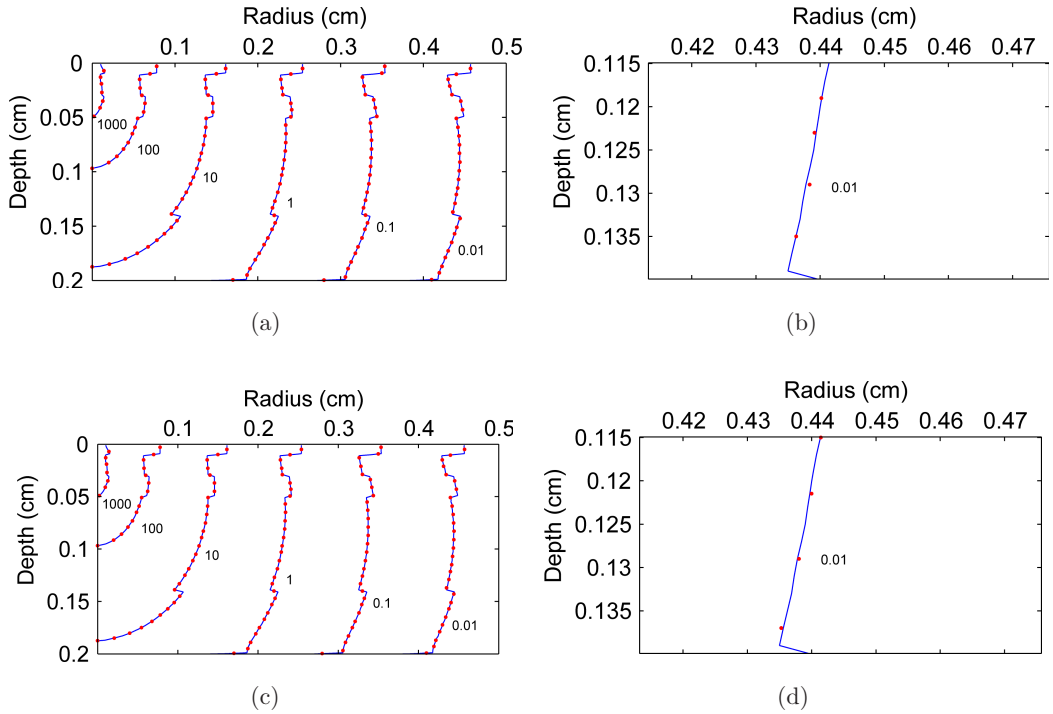


Fig. 5. Distribution of the internal absorption in form of contours and the values for the contours are 1000, 100, 10, 1, 0.1 and 0.01 from left to right, respectively. (a) The contours of GCMCML simulation (red dots) and MCML simulation (blue line). (b) The enlarged picture of the part of figure (a) for the last contour. (c) The contours of two MCML simulations with different random sequences. (d) The enlarged picture of the part of figure (c) for the last contour.

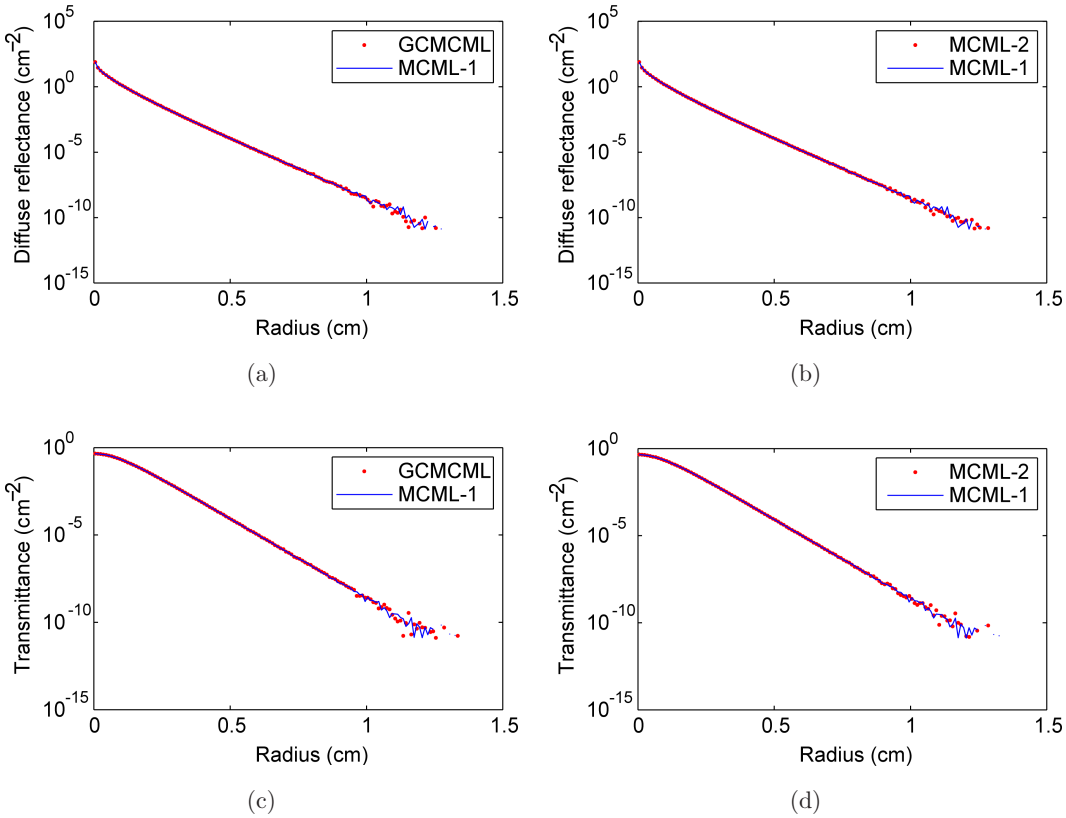


Fig. 6. Distribution of diffuse reflectance and transmittance along the radius. (a) The diffuse reflectance of GCMCML simulation (red dots) and MCML simulation (blue line). (b) The diffuse reflectance of the two MCML simulations with different random sequences. (c) The transmittance of GCMCML simulation (red dots) and MCML simulation (blue line). (d) The transmittance of the two MCML simulations with different random sequences.

sequences, rather than the errors induced by GPU-based implementations.

Figure 6 shows the comparisons of the diffuse reflectance distribution and the transmittance distribution along the radius. As shown in Figs. 6(a) and 6(c), the results of the two MC programs fit each other well in total. When the radius is roughly smaller than 1 cm, the relative difference can hardly be told. But when the radius is roughly larger than 1 cm, the difference begins to increase. These differences can also be explained by the statistical uncertainty between runs due to the different random sequences, since the comparisons between two CPU-based MCML simulations with different random sequences show the similar relative differences, as illustrated in Figs. 6(b) and 6(d).

Table 3 shows the total diffuse reflectance and the total transmittance of GCMCML simulation and the two MCML simulations with different random sequences. The relative differences to one of the

Table 3. The total diffuse reflectance and transmittance of different runs with different MC programs and different random sequences.

Version	Diffuse reflectance	Relative difference	Transmittance	Relative difference
MCML-1	0.240877	0	0.0214835	0
MCML-2	0.240884	2.91×10^{-5}	0.0214938	4.794×10^{-4}
MCML-3	0.240901	9.96×10^{-5}	0.0214897	2.886×10^{-4}
MCML-4	0.24084	1.536×10^{-4}	0.0214924	4.143×10^{-4}
MCML-5	0.240815	2.574×10^{-4}	0.0214905	3.258×10^{-4}
GCMCML	0.240909	1.328×10^{-4}	0.0214943	5.027×10^{-4}

five MCML simulation results (MCML-1) are also presented. The relative differences for the GCMCML and MCML are roughly within the same scope (about $10^{-5} \sim 10^{-4}$) as those of the five MCML simulations with different random sequences. This further indicates that the relative differences are induced by the different random sequences rather than GPU.

5. Conclusion and Discussion

In this paper, a GPU cluster-based Monte Carlo simulation method is presented. By employing network distributed computing and GPU, the performance is greatly improved in contrast to the CPU-based method and the single GPU-based Monte Carlo method.

When the GPUs of the cluster are of the same type, our GCMCML runs roughly N times faster than using single GPU-based simulation, where N is the total number of the GPUs within the cluster. If the GPUs of different types are added into the same cluster, the speedup will no longer be the total number of the GPUs. This is because the main node allocates the simulation task on average for all the nodes in current program and the GPUs of different types have different processing capability. The poorest GPU of the cluster will be the bottleneck of the total performance. Therefore, a more optimized method for allocating simulation tasks considering the processing capability of GPUs rather than simply the total number of GPUs is needed. We will work on for this GPU cluster-based MCML to solve this problem.

Acknowledgments

This work is supported by the program for New Century Excellent Talents in University (Grant No. NCET-08-0213), Science Fund for Creative Research Group of China (Grant No. 61121004), the National Natural Science Foundation of China (Grant Nos. 30970964, 30800339, 30801482, and 30800313), and the Ph.D. Programs Foundation of Ministry of Education of China (Grant No. 20090142110054).

References

1. B. C. Wilson, G. Adam, "A Monte Carlo model for the absorption and flux distributions of light in tissue," *Med. Phys.* **10**(6), 824 (1983).
2. L. Wang, S. L. Jacques, L. Zheng, "MCML — Monte Carlo modeling of light transport in multi-layered tissues," *Comput. Meth. Programs Biomed.* **47**(2), 131 (1995).
3. D. R. Kirkby, D. T. Delpy, "Parallel operation of Monte Carlo simulations on a diverse network of computers," *Phys. Med. Biol.* **42**(6), 1203 (1997).
4. A. Colasanti, G. Guida, A. Kisslinger, R. Liuzzi, M. Quarto, P. Riccio, G. Roberti, F. Villani, "Multiple processor version of a Monte Carlo code for photon transport in turbid media," *Comput. Phys. Commun.* **132**(1−2), 84 (2000).
5. W. C. Y. Lo, K. Redmond, J. Luu, P. Chow, J. Rose, L. Lilge, "Hardware acceleration of a Monte Carlo simulation for photodynamic treatment planning," *J. Biomed. Opt.* **14**(1), 014019 (2009).
6. E. Alerstam, T. Svensson, S. Andersson-Engels, "Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration," *J. Biomed. Opt.* **13**(6), 060504 (2008).
7. E. Alerstam, T. Svensson, S. Andersson-Engels, CUDAMCML User manual and implementation notes (2009).
8. W. C. Y. Lo, T. D. Han, J. Rose, L. Lilge, GPU-accelerated Monte Carlo simulation for photodynamic therapy treatment planning, *Proc. SPIE*, Vol. 7373, pp. 737313 (2009).
9. E. Alerstam, W. Lo, C. Yip, T. Han, D. Rose, J. S. Andersson-Engels, L. Lilge, "Next-generation acceleration and code optimization for light transport in turbid media using GPUs," *Biomed. Opt. Express* **1**(2), 658 (2010).
10. Q. Fang, D. A. Boas, "Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units," *Opt. Express* **17**(22), 20178 (2009).
11. N. Ren, J. Liang, X. Qu, J. Li, B. Lu, J. Tian, "GPU-based Monte Carlo simulation for light propagation in complex heterogeneous tissues," *Opt. Express* **18**(7), 6811 (2010).
12. C. Nvidia, Compute Unified Device Architecture, Programming Guide, version 1.1 (2007).
13. G. Marsaglia, "Random number generators," *J. Mod. Appl. Stat. Meth.* **2**(1), 2 (2003).
14. W. Gropp, E. Lusk, N. Doss, A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.* **22**(6), 789 (1996).
15. V. V. Tuchin, "Light scattering study of tissues," *Phys. Usp.* **40**(5), 495 (1997).