

Development of a GPU-based Monte Carlo dose calculation code for coupled electron–photon transport

This article has been downloaded from IOPscience. Please scroll down to see the full text article.

2010 Phys. Med. Biol. 55 3077

(<http://iopscience.iop.org/0031-9155/55/11/006>)

View [the table of contents for this issue](#), or go to the [journal homepage](#) for more

Download details:

IP Address: 137.110.121.31

The article was downloaded on 13/05/2010 at 06:25

Please note that [terms and conditions apply](#).

Development of a GPU-based Monte Carlo dose calculation code for coupled electron–photon transport

Xun Jia¹, Xuejun Gu¹, Josep Sempau², Dongju Choi³,
Amitava Majumdar³ and Steve B Jiang¹

¹ Department of Radiation Oncology, University of California San Diego, La Jolla, CA 92037-0843, USA

² Institut de Techniques Energetiques, Universitat Politecnica de Catalunya, Diagonal 647, E-08028, Barcelona, Spain

³ San Diego Supercomputer Center, University of California San Diego, La Jolla, CA 92093, USA

E-mail: sbjiang@ucsd.edu

Received 28 October 2009, in final form 22 March 2010

Published 12 May 2010

Online at stacks.iop.org/PMB/55/3077

Abstract

Monte Carlo simulation is the most accurate method for absorbed dose calculations in radiotherapy. Its efficiency still requires improvement for routine clinical applications, especially for online adaptive radiotherapy. In this paper, we report our recent development on a GPU-based Monte Carlo dose calculation code for coupled electron–photon transport. We have implemented the dose planning method (DPM) Monte Carlo dose calculation package (Sempau *et al* 2000 *Phys. Med. Biol.* **45** 2263–91) on the GPU architecture under the CUDA platform. The implementation has been tested with respect to the original sequential DPM code on the CPU in phantoms with water–lung–water or water–bone–water slab geometry. A 20 MeV mono-energetic electron point source or a 6 MV photon point source is used in our validation. The results demonstrate adequate accuracy of our GPU implementation for both electron and photon beams in the radiotherapy energy range. Speed-up factors of about 5.0–6.6 times have been observed, using an NVIDIA Tesla C1060 GPU card against a 2.27 GHz Intel Xeon CPU processor.

(Some figures in this article are in colour only in the electronic version)

1. Introduction

Dose calculation is of central importance in radiotherapy treatment planning. Among algorithms developed for solving this problem, Monte Carlo (MC) simulation is considered to be the most accurate method. A number of software packages have been developed for MC simulation of radiation transport, ranging from those packages for general purposes, such as EGS4 (Nelson *et al* 1985, Bielajew *et al* 1994), EGSnrc (Kawrakow 2000), MCNP (Briesmeister 1993), PENELOPE (Baró *et al* 1995, Salvat *et al* 1996, 2009, Sempau

et al 1997), GEANT4 (Agostinelli *et al* 2003), to those clinically oriented ones such as VMC++ (Kawrakow *et al* 1996), MCDOSE/MCSIM (Ma *et al* 1999, 2002, Li *et al* 2000) and dose planning method (DPM) (Sempau *et al* 2000), to name a few.

Since MC simulation is a statistical method, its accuracy largely depends on the number of simulated particle histories. In MC simulation, we compute, from first principles, how a particle evolves step by step. A large number of particle histories are simulated in order to achieve the desired statistical accuracy. Therefore, despite the vast development in computer architecture and increase in processor clock speed in recent years, the efficiency of the currently available full MC dose engines is still not completely satisfactory for routine clinical applications in radiotherapy treatment planning.

One way to overcome this obstacle is to perform the computation in a parallel fashion by taking advantage of advanced computer architectures, such as CPU clusters or general-purpose graphics processing units (GPUs). Compared to CPU clusters of similar parallel computing power, the GPU is easier to access and maintain (it can run on a local desktop computer) and much less expensive (one to two orders of magnitude lower cost). With affordable graphic cards such as NVIDIA's GeForce, GTX and Tesla series, GPU-based computing has recently been utilized to speed up heavy duty computational tasks in radiotherapy, such as cone-beam CT reconstruction (Xu and Mueller 2005, 2007, Li *et al* 2007, Sharp *et al* 2007, Yan *et al* 2008, Jia *et al* 2010), deformable image registration (Sharp *et al* 2007, Samant *et al* 2008, Gu *et al* 2010), dose calculation (Jacques *et al* 2008, Hissoiny *et al* 2009, Gu *et al* 2009) and treatment plan optimization (Men *et al* 2009). In particular, Jacques *et al* (2008) and Hissoiny *et al* (2009) have explored GPUs for fast dose computation via the superposition/convolution algorithm and Gu *et al* (2009) implemented a finite size pencil beam model for GPU-based dose calculation.

In this paper, we report our recent development of a Monte Carlo dose calculation package under the compute unified device architecture (CUDA) platform developed by NVIDIA (2009), which enables us to extend C language to program GPU. At University of California San Diego (UCSD), we have implemented the full MC dose calculation package DPM with coupled electron–photon transport based on reasonably priced and readily available GPUs. Our code is benchmarked against the original sequential DPM package on the CPU. This paper is organized as follows. In section 2, we describe the general structure of the DPM package and some key issues of our implementation. Section 3 presents experimental results of our dose calculation in heterogeneous phantoms. The computational time, as well as simulation results, is compared between GPU and CPU implementations. Finally, we conclude in section 4 and discuss possible limitations of our GPU implementation.

2. Methods and materials

2.1. DPM MC simulation

The DPM Monte Carlo code has been developed for dose calculations in radiotherapy treatment planning (Sempau *et al* 2000). It employs an accurate and efficient coupled electron–photon transport scheme and a set of significant innovations for transporting particles in order to gain performance. The electron transport part of the DPM relies on a condensed history technique for elastic collisions. Detailed (i.e. step-by-step) simulation is used for inelastic collisions and bremsstrahlung emission involving energy losses above certain cutoffs. Below these cutoffs, the continuous slowing down approximation (CSDA) is employed. As for the photon transport, the DPM implements the Woodcock tracking method, which significantly increases the simulation efficiency of the boundary tracking process (Woodcock *et al* 1965). The DPM

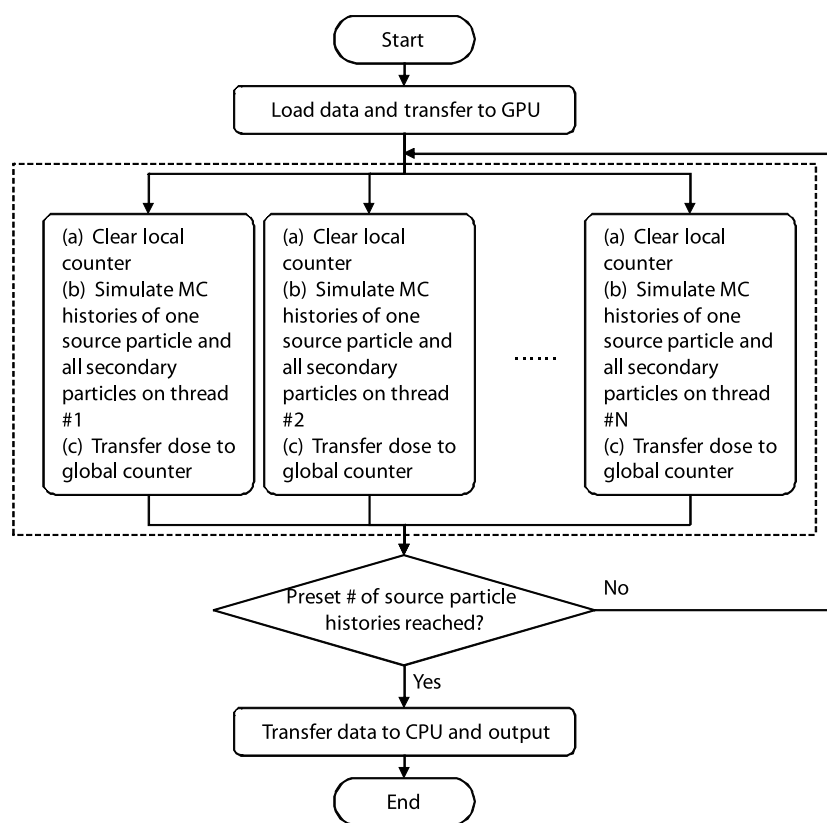


Figure 1. The flow chart of our GPU-based DPM Monte Carlo simulation. The area bounded by dashed lines indicates the kernel simulating histories of one source particle as well as all secondary particles. The kernel is executed on the GPU N times with N different CUDA threads in parallel.

also focuses on a small dynamic range (in energy and material) of radiotherapy problems, where some approximations are valid, so as to speed up the calculation. The accuracy of the DPM has been demonstrated to be within $\pm 2\%$ for both clinical photon and electron beams (Chetty *et al* 2002, 2003).

2.2. CUDA implementation

Our GPU-based DPM code is developed using CUDA as programming environment and NVIDIA GPU cards as the hardware platform. Our implementation is an exact translation of the original sequential DPM code into a parallel version on the GPU. Specifically, we treat each computational thread on the GPU as an independent computing unit, which tracks complete histories of a source particle as well as all secondary particles it generates, as if in the original sequential DPM code. This multiple-thread simulation is performed in a batched fashion. A large number of CUDA threads run simultaneously in every batch, and therefore, efficiency can be gained due to this vast parallelization.

This computation procedure is schematically depicted in figure 1. After initializing the program, we first load all the relevant data from files on the hard disk to GPU global memory. For example, cross section data are prepared as a function of energy with respect to all physical interactions and materials. Random seeds are generated on the CPU and then passed to the

GPU in this step as well. After this preparation stage, a special C function, termed *kernel*, is invoked by the CPU and is executed N times on a total number of N threads in a parallel manner on the GPU, where each thread performs the calculation independently, as illustrated by the big dash box in figure 1. There are three steps within each thread. (a) *Local counter initialization*: a local counter, an array privately owned by a thread for scoring the dose deposition from the particle histories followed by the thread, is initialized. (b) *Particle simulation*: in this step, an empty stack is first initialized. A source particle is generated according to a source model. Simulation with respect to this particle is then performed and dose information is stored on the local counter whenever dose deposition occurs. In this process, any secondary particles due to some physical interactions, e.g. Compton scattering, are recorded in the stack. Upon completing the simulation for the current particle, another particle is fetched from the stack and simulation with respect to the particle is performed. Such a process is repeated till the stack becomes empty. (c) *Dose collection*: the dose deposition obtained from each thread is transferred onto a global dose counter and summation of the dose deposition is performed where necessary. Such a kernel is executed repeatedly till a preset number of histories is achieved. Finally, the dose on the global counter is transferred from GPU memory to CPU memory and output before the entire program terminates.

2.3. Data structure

GPU data structure is an important issue in the MC simulation. Since double-precision arithmetic is not fully supported in CUDA, throughout our implementation, single-precision floating point data type is used to represent rational numbers instead of double-precision as in the original sequential DPM code. This, however, does not seem to reduce the simulation accuracy, as will be seen in the results presented below.

In our simulation, variables are stored differently in GPU memory depending on how they are accessed. First, read-only data are those accessed by each thread frequently during the simulation without modification. It is desirable to store data of this type in the constant memory of the GPU, which provides a high access speed. However, only those variables of small sizes are stored in this way due to the limitation of constant memory space of the GPU (~64 k). As for read-only data of large sizes, we store them on GPU global memory in the form of texture memory. Since accessing texture memory is cached, high performance speed can be achieved during the simulation. Examples in this category are the cross-section data for all types of collisions as a function of energy in each material. Finally, those data constantly being modified are stored in the global memory of the GPU. For instance, it is necessary for each thread to keep and update its own random number seeds in order to generate independent random number sequences. The local dose counter mentioned above also belongs to this category, which is frequently modified to record the dose deposition for the particle histories tracked by the particular thread only. The use of this data structure, however, has two limitations on the performance of our code. In the first place, there is a huge demand of GPU memory in order to store the variables of this kind, since most of them have to be allocated multiple times, one for each thread, during the simulation. As a result, the number of threads is restricted by the total amount of GPU memory, which limits the capability of the parallelization. Second, global memory is accessed in an un-cached way and thus using it is very time consuming.

2.4. Other technical issues

One technical detail worth addressing here is how the dose deposition is recorded in our simulation. Within the MC simulation kernel, each thread updates its own local counter every

time energy is deposited in a voxel. Since it is not possible to predict where a dose deposition will occur, the simplest approach seems to establish a one-to-one correspondence between voxels and the elements of the local counter array, so that each element keeps tracking the energy deposited in the corresponding voxel. Nonetheless, neither is this strategy memory efficient nor necessary. Indeed, the energy is only deposited in a few number of voxels within histories of a source particle and all subsequent particles it generates, as these particles only travels through a small fraction of total voxels. It is therefore feasible to allocate the local dose counter with a predetermined length, which records the amount of energy being deposited and the location where the corresponding deposition occurs. The length of this local counter has to be carefully chosen to be long enough.

Upon exiting the kernel, all threads transfer dose deposition from their own local counters to the global counter. In this process, energy deposited in a particular voxel is summed over different threads. This, however, potentially causes a memory conflict due to the attempt of writing to a same memory address from different threads. We adopt an atomic float addition function developed by Lensch and Strzodka (2008) to resolve this problem. This function is atomic in the sense that, once one thread is writing to a memory address during the summation, no other threads can interfere this process.

Another key issue in MC simulation is the random number generator, whose quality plays an important role in the simulation. In our implementation, we take the pseudo-random number generator RANECU (L'Ecuyer 1988), which has a total period of $\sim 2 \times 10^{18}$. In addition, different CUDA threads are initialized with different random seeds generated by a CPU random number generator. During the simulation, each CUDA thread deals with its own random seeds, so that the random number sequences produced from distinct threads are statistically independent.

2.5. GPU card

Computer graphic cards, such as the NVIDIA GeForce series and the GTX series, are conventionally used for display purpose on desktop computers. They typically consist of 32–240 scalar processor units and 256 MB to 1 GB memory. Recently, NVIDIA introduced special GPUs solely dedicated for scientific computing, for example the Tesla C1060 card that is used in our MC simulation. Such a GPU card has a total number of 240 processor cores (grouped into 30 multiprocessors with 8 cores each), each with a clock speed of 1.3 GHz. The card is equipped with 4 GB DDR3 memory, shared by all processor cores.

3. Experimental results

We investigated the performance of our GPU-based DPM MC simulation on various phantom experiments. Two testing phantoms are considered in our simulation, both of which are $30.5 \times 30.5 \times 30.0 \text{ cm}^3$ rectangular cuboids with a voxel size of $0.5 \times 0.5 \times 0.2 \text{ cm}^3$. Along the z direction, the phantoms consist of three layers, namely either water–bone–water layers of thicknesses 5 cm, 5 cm and 20 cm, respectively, or water–lung–water layers of the same dimensions. Either an electron or a photon point beam sources with $SSD = 90 \text{ cm}$ impinges normally on the center of the phantoms at the x – o – y plane. The electron beam is chosen to be 20 MeV mono-energetic, while the photon beam is generated according to a realistic 6 MV energy spectrum. For both the electron and the photon beams, field sizes are set to $10 \times 10 \text{ cm}^2$ at the isocenter with $SAD = 100 \text{ cm}$. The absorption energies are 200 keV for electron and 50 keV for photon. A total number of 10^7 and 10^9 particle histories are simulated for the electron source and the photon source cases, respectively. The statistical uncertainty of

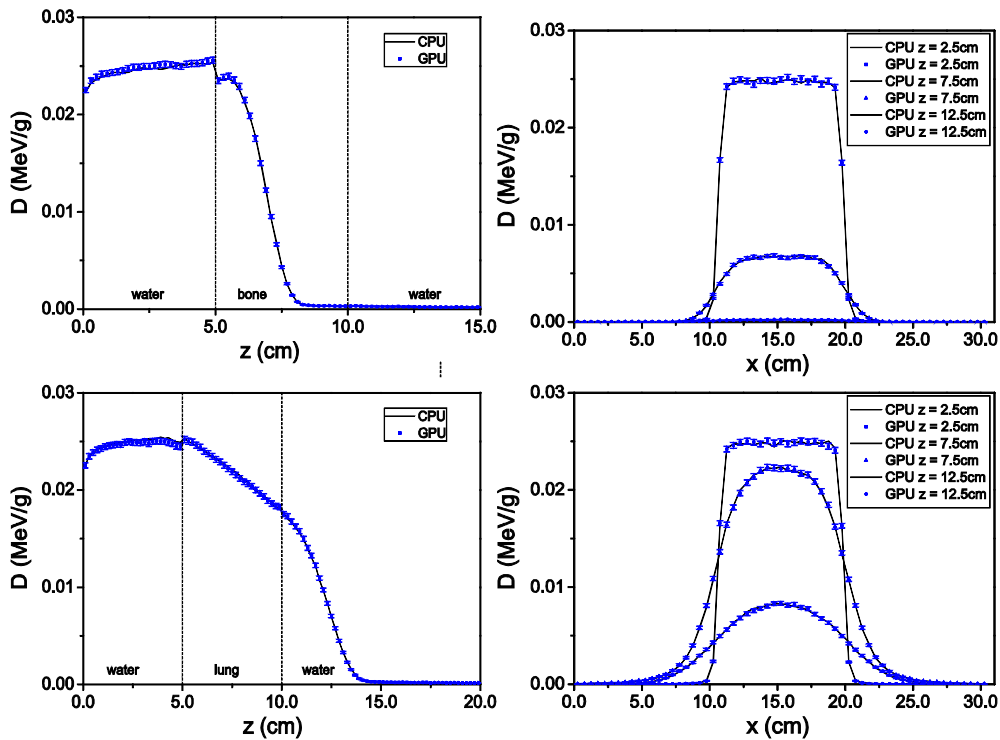


Figure 2. Depth-dose curves (left column) and lateral dose profiles at different depths (right column) of a $10 \times 10 \text{ cm}^2$, 20 MeV electron point source at $\text{SSD} = 90 \text{ cm}$ impinging on a water–bone–water phantom (top row) and a water–lung–water phantom (bottom row). Error bars correspond to 2 standard deviations.

our simulation is characterized by the averaged relative uncertainties $\overline{\sigma_D/D}$, where σ_D is the standard deviation of the local dose D at a voxel. The over bar stands for an average over the region where the local dose D exceeds half of its maximum value in the entire phantom. The number of particle histories simulated is large enough, so that $\overline{\sigma_D/D}$ were found to be less than 1% in all cases studied. In order to validate our GPU code and test its efficiency, we also run the original sequential DPM code on a CPU and used the results as reference for comparison. The original DPM code is executed on a 2.27 GHz Intel Xeon processor, while the GPU code on a NVIDIA Tesla C1060 card.

Figures 2 and 3 demonstrate the dose distribution for the tested electron and photon beam cases, respectively. In both figures, the left columns are the depth dose curves along the central axis, while the right columns correspond to the lateral dose profiles at different depths. The simulation results for the water–bone–water phantom and the water–lung–water phantom are shown in the top rows and the bottom rows, respectively. The error bars represent $2\sigma_D$ levels of the results. For the purpose of clarity, we did not draw error bars for the results obtained from the CPU code, which are of similar sizes to those for the results from the GPU. Results obtained from the CPU and the GPU are found to be in good agreement with each other. The difference at more than 98% of the calculation points is within 1% of the maximum dose all testing cases. In table 1, we list the average relative uncertainties $\overline{\sigma_D/D}$ in all testing cases, which are controlled to be less than 1% in our calculation.

Table 1 also depicts the computation time in the two testing cases. T_{CPU} stands for the execution time of the CPU implementation, while T_{GPU} is that of the GPU implementation. In

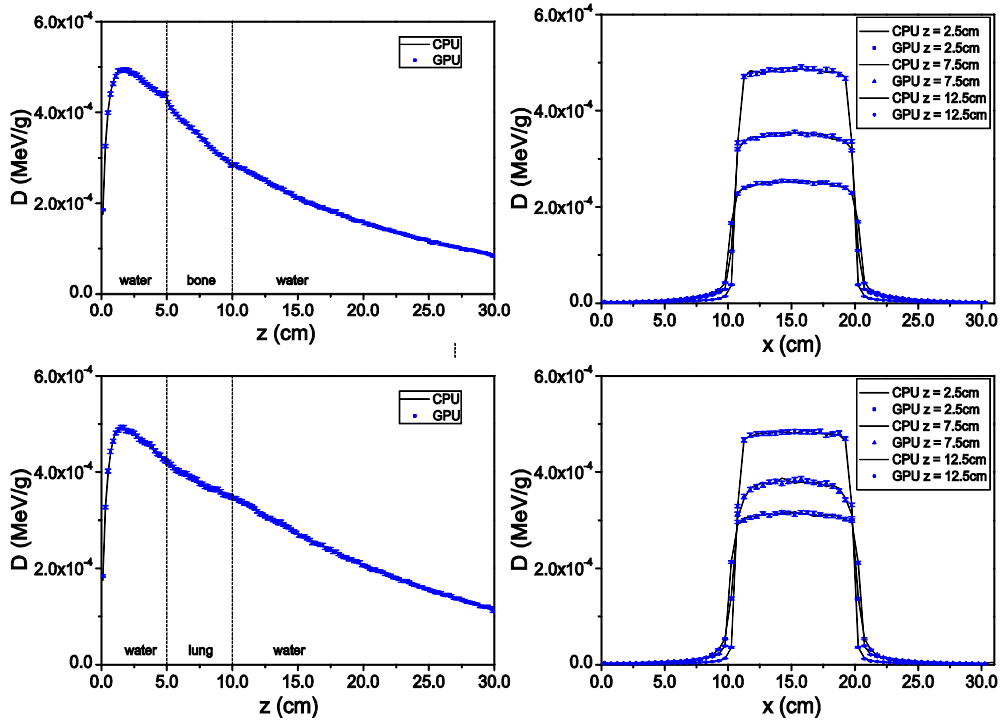


Figure 3. Depth–dose curves (left column) and lateral dose profiles at different depths (right column) of a 10×10 cm², 6 MV photon point source at SSD = 90 cm impinging on a water–bone–water phantom (top row) and a water–lung–water phantom (bottom row). Error bars correspond to 2 standard deviations.

Table 1. Average relative uncertainty ($\frac{\sigma_D}{D}$) execution time (T), and speed-up factors ($\frac{T_{\text{CPU}}}{T_{\text{GPU}}}$) for four different testing cases.

| Source type | No of histories | Phantom | $\overline{\sigma_D/D}$ CPU (%) | $\overline{\sigma_D/D}$ GPU (%) | T_{CPU} (s) | T_{GPU} (s) | $\frac{T_{\text{CPU}}}{T_{\text{GPU}}}$ |
|-----------------|-----------------|------------------|---------------------------------|---------------------------------|----------------------|----------------------|---|
| 20 MeV electron | 10^7 | Water–lung–water | 0.65 | 0.65 | 470.0 | 80.6 | 5.83 |
| 20 MeV electron | 10^7 | Water–bone–water | 0.63 | 0.63 | 508.0 | 101.8 | 4.99 |
| 6 MV photon | 10^9 | Water–lung–water | 0.51 | 0.52 | 5615.0 | 845.8 | 6.63 |
| 6 MV photon | 10^9 | Water–bone–water | 0.45 | 0.46 | 6964.0 | 1242.3 | 5.61 |

all time measurements, the time T correspond to the complete MC simulation time, including the time used for program initialization, finalization and data transferring from the CPU to the GPU in the GPU implementation. Speed-up factors of about 5.0–6.6 times have been observed in the GPU calculation compared to CPU simulation. These speed-up factors on the GPU are achieved using 100 CUDA blocks and 128 threads per block. These numbers regarding GPU configurations are carefully tuned in order to achieve the best performance.

4. Discussion and conclusions

In this paper, we have successfully implemented the DPM Monte Carlo dose calculation package on GPU architecture under the NVIDIA CUDA platform. We have also tested the

efficiency and accuracy of our GPU implementation with respect to the original sequential DPM code on the CPU in various testing cases. Our results demonstrate the adequate accuracy of our implementation for both electron and photon sources. Speed-up factors of about 5.0–6.6 times have been observed. The code is in public domain and available to readers on request.

MC simulations are known as embarrassingly parallel because they are readily adaptable for parallel computing. It has been reported that the DPM package has been parallelized on a CPU cluster, and roughly a linear increase of speed has been realized with respect to an increasing number of processors when up to 32 nodes of an Intel cluster are involved (Tyagi *et al* 2004). Nonetheless, this linear scalability is hard to achieve on GPU architectures, when simply distributing particles to all threads and treating them as if they were independent computational units.

In general, the means of performing parallel computation are categorized into *task parallelization* and *data parallelization*. MC simulation, a typical task parallelization problem, is preferable for a CPU cluster developed through, for example, *message passing interface* (MPI). All particles histories simulated in an MC dose calculation can be distributed to all processors, which execute simultaneously without interfering with each other. Only at the end of the computation will the dose distribution need to be collected from all processors. Apparently the parallelization of this manner is capable of speeding up the simulation easily with a large number of CPU nodes. On the other hand, the GPU is known to be suitable for the data parallelization problems. A GPU multiprocessor employs an architecture called *SIMT* (single-instruction, multiple-thread) (NVIDIA 2009). Under such architecture, the multiprocessor executes program in groups of 32 parallel threads termed *warps*. If the paths for threads within a warp diverge due to, e.g., some *if-else* statements, the warp serially executes one thread at a time while putting all other threads in an idle state. Thus, high computation efficiency is only achieved when 32 threads in a warp process together along the same execution path. Unfortunately, in an MC calculation, the computational work paths on different threads are statistically independent, essentially resulting in a serial execution within a warp. Since there are physically only 30 multiprocessors in a GPU, our simulation is indeed parallelized by just 30 independent computation units. Considering, furthermore, that the GPU clock speed is 1.3 GHz, about a half of that for the CPU, the highest possible speed-up factor for a GPU-based MC simulation will be roughly 15 times.

Other factors may also adversely restrict our simulation efficiency, such as the memory access pattern. Since all threads share the usage of a global memory in our code, the random access to different memory addresses from different threads produces a serious overhead. In addition, the global memory on the GPU is not cached, leading to 400–600 clock cycles of memory latency, while the CPU memory is normally cached and favorable for fast fetching. In a nut shell, due to many factors limiting our simulation efficiency, our code has only achieved speed-up about 5.0–6.6 times.

On the other hand, the GPU implementation of an MC simulation still has the obvious advantage with its low cost and easy accessibility as opposed to the CPU clusters. Our work has clearly demonstrated the potential possibility of speeding up an MC simulation with the help of the GPU. Currently, another MC simulation algorithm which is specifically tailored for GPU architecture is under development and a boost of MC simulation speed is expected.

Acknowledgments

This work is supported in part by the University of California Lab Fees Research Program. We would like to thank NVIDIA for providing GPU cards for this project. JS acknowledges

partial financial support from the Spanish Ministerio de Educacion y Ciencia, project no FIS2006-07016.

References

- Agostinelli S *et al* 2003 GEANT4—a simulation toolkit *Nucl. Instrum. Methods Phys. Res A* **506** 250–303
- Baró J, Sempau J, Fernández-Varea J M and Salvat F 1995 PENELOPE: an algorithm for Monte Carlo simulation of the penetration and energy loss of electrons and positrons in matter *Nucl. Instrum. Methods B* **100** 31–46
- Bielajew A F, Hirayama H, Nelson W R and Rogers D W O 1994 History, overview and recent improvements of EGS4 *National Research Council of Canada Report PIRS-0436*
- Briesmeister J F 1993 MCNP—a general Monte Carlo N-particle transport code *Los Alamos National Laboratory Report LA-12625-M*
- Chetty I J *et al* 2003 Photon beam relative dose validation of the DPM Monte Carlo code in lung-equivalent media *Med. Phys.* **30** 563–73
- Chetty I J *et al* 2002 Benchmarking of the dose planning method (DPM) Monte Carlo code using electron beams from a racetrack microtron *Med. Phys.* **29** 1035–41
- Gu X *et al* 2009 GPU-based ultra fast dose calculation using a finite size pencil beam model *Phys. Med. Biol.* **54** 6287–97
- Gu X *et al* 2010 Implementation and evaluation of various demons deformable image registration algorithms on a GPU *Phys. Med. Biol.* **55** 207–19
- Hissoiny S, Ozell B and Després P 2009 Fast convolution-superposition dose calculation on graphics hardware *Med. Phys.* **36** 1998–2005
- Jacques R, Taylor R, Wong J and McNutt T 2008 Towards real-time radiation therapy: GPU accelerated superposition/convolution *High-Performance Medical Image Computing and Computer Aided Intervention Workshop*
- Jia X, Lou Y, Li R, Song W Y and Jiang S B 2010 GPU-based fast cone beam CT reconstruction from undersampled and noisy projection data via total variation *Med. Phys.* **37** 1757–60
- Kawrakow I 2000 Accurate condensed history Monte Carlo simulation of electron transport: I. EGSnrc, the new EGS4 version *Med. Phys.* **27** 485–98
- Kawrakow I, Fippel M and Friedrich K 1996 3D electron dose calculation using a voxel based Monte Carlo algorithm (VMC) *Med. Phys.* **23** 445–57
- L’Ecuyer P 1988 Efficient and portable combined random number generators *Comm. ACM* **31** 742–9
- Lensch H and Strzodka R 2008 Massively parallel computing with CUDA <http://www.mpi-inf.mpg.de/~strzodka/lectures/ParCo08/>
- Li J S *et al* 2000 Validation of a Monte Carlo dose calculation tool for radiotherapy treatment planning *Phys. Med. Biol.* **45** 2969–85
- Li M, Yang H, Koizumi K and Kudo H 2007 Fast cone-beam CT reconstruction using CUDA architecture *Med. Imaging Technol.* **25** 243–50
- Ma C M *et al* 2002 A Monte Carlo dose calculation tool for radiotherapy treatment planning *Phys. Med. Biol.* **47** 1671–89
- Ma C-M *et al* 1999 Clinical implementation of a Monte Carlo treatment planning system *Med. Phys.* **26** 2133–43
- Men C *et al* 2009 GPU-based ultra fast IMRT plan optimization *Phys. Med. Biol.* **54** 6565–73
- Nelson W R, Hirayama H and Rogers D W O 1985 The EGS4 code system *Stanford Linear Accelerator Center Report SLAC-265*
- NVIDIA 2009 *NVIDIA CUDA Compute Unified Device Architecture, Programming Guide*
- Salvat F, Fernández-Varea J M, Baró J and Sempau J 1996 PENELOPE, an algorithm and computer code for Monte Carlo simulation of electron-photon showers *Technical Report no 799* (Madrid: CIEMAT)
- Salvat F, Fernández-Varea J M and Sempau J 2009 PENELOPE-2008: a code system for Monte Carlo simulation of electron and photon transport (Issy-les-Moulineaux: OECD-NEA)
- Samant S S, Xia J Y, Muyan-Ozcelik P and Owens J D 2008 High performance computing for deformable image registration: towards a new paradigm in adaptive radiotherapy *Med. Phys.* **35** 3546–53
- Sempau J, Acosta E, Baró J, Fernández-Varea J M and Salvat F 1997 An algorithm for Monte Carlo simulation of coupled electron-photon showers *Nucl. Instrum. Methods B* **132** 377–90
- Sempau J, Wilderman S J and Bielajew A F 2000 DPM, a fast, accurate Monte Carlo code optimized for photon and electron radiotherapy treatment planning dose calculations *Phys. Med. Biol.* **45** 2263–91
- Sharp G C, Kandasamy N, Singh H and Folkert M 2007 GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration *Phys. Med. Biol.* **52** 5771–83

- Tyagi N, Bose A and Chetty I J 2004 Implementation of the DPM Monte Carlo code on a parallel architecture for treatment planning applications *Med. Phys.* **31** 2721–5
- Woodcock E, Murphy T, Hemmings P and Longworth S 1965 Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry *Applications of Computing Methods to Reactor Problems: Argonne National Laboratories Report ANL-7050*
- Xu F and Mueller K 2005 Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware *IEEE Trans. Nucl. Sci.* **52** 654–63
- Xu F and Mueller K 2007 Real-time 3D computed tomographic reconstruction using commodity graphics hardware *Phys. Med. Biol.* **52** 3405–19
- Yan G R, Tian J, Zhu S P, Dai Y K and Qin C H 2008 Fast cone-beam CT image reconstruction using GPU hardware *J. X-Ray Sci. Technol.* **16** 225–34