Annals of Nuclear Energy 82 (2015) 230–239

Contents lists available at ScienceDirect

Annals of Nuclear Energy

journal homepage: www.elsevier.com/locate/anucene

Comparison of two accelerators for Monte Carlo radiation transport calculations, Nvidia Tesla M2090 GPU and Intel Xeon Phi 5110p coprocessor: A case study for X-ray CT imaging dose calculation

T. Liu [a], X.G. Xu [a,*], C.D. Carothers [b]

[a] Nuclear Engineering Program, Rensselaer Polytechnic Institute (RPI), Troy, NY 12180, United States
[b] Computer Science Department, Rensselaer Polytechnic Institute (RPI), United States

ARTICLE INFO

Article history:
Received 30 April 2014
Accepted 26 August 2014
Available online 29 September 2014

Keywords:
Monte Carlo
Nvidia GPU
Fermi
Intel Xeon Phi coprocessor
MIC
CT imaging dose

ABSTRACT

Hardware accelerators are currently becoming increasingly important in boosting high performance computing systems. In this study, we tested the performance of two accelerator models, Nvidia Tesla M2090 GPU and Intel Xeon Phi 5110p coprocessor, using a new Monte Carlo photon transport package called ARCHER-CT we have developed for fast CT imaging dose calculation. The package contains three components, ARCHER-CT$_{CPU}$, ARCHER-CT$_{GPU}$ and ARCHER-CT$_{COP}$ designed to be run on the multi-core CPU, GPU and coprocessor architectures respectively. A detailed GE LightSpeed Multi-Detector Computed Tomography (MDCT) scanner model and a family of voxel patient phantoms are included in the code to calculate absorbed dose to radiosensitive organs under user-specified scan protocols. The results from ARCHER agree well with those from the production code Monte Carlo N-Particle eXtended (MCNPX). It is found that all the code components are significantly faster than the parallel MCNPX run on 12 MPI processes, and that the GPU and coprocessor codes are 5.15–5.81 and 3.30–3.38 times faster than the parallel ARCHER-CT$_{CPU}$, respectively. The M2090 GPU performs better than the 5110p coprocessor in our specific test. Besides, the heterogeneous computation mode in which the CPU and the hardware accelerator work concurrently can increase the overall performance by 13–18%.

1. Introduction

In recent years, there has been a substantial increase in the use of heterogeneous systems for high performance computing. The traditional CPUs are accompanied by the hardware accelerators, which are well known for their high energy efficiency (Keckler et al., 2011), to largely increase the computing power. The compute-intensive and parallelizable tasks can be offloaded through the Peripheral Component Interconnect Express (PCIe) to the plug-and-play accelerators, such as the Graphics Processing Unit (GPUs) and the Many Integrated Core (MIC) coprocessors, for efficient parallel execution. By June 2014, Nvidia GPUs have been utilized in 46 most powerful supercomputer systems worldwide on the top-500 list and Intel coprocessors in 19 systems (Top500, 2014). Among them, the No. 1 supercomputer Milky Way-2 developed by China contains 48,000 Intel Xeon Phi 31S1P MIC coprocessors (57 cores per coprocessor), and the No. 2 Titan developed by Oak Ridge National Laboratory contains 18,688 Nvidia K20x GPUs (14 Streaming Multiprocessors per GPU).

Since the GPUs with high-level programming language support were first introduced by Nvidia in 2006, several groups have applied them to Monte Carlo (MC) photon and electron transport simulations. Badal and Badano (2009, 2011) developed the MC-GPU code for X-ray radiography simulation and radiography dose calculation, and reported a speedup of 110 over the CPU-based MC code, PENELOPE (Baró et al., 1995). Jia et al. (2010, 2011) developed the gDPM code based on the CPU code DPM originally written by Sempau et al. (2000), and observed a speedup of 69.1–87.2 over the CPU code for radiotherapy dose calculations. Hissoiny et al. (2011) developed the GPUMCD code for coupled electron-photon transport and reported that for electron transport the speedup factors were 210 and 1200 compared to the general-purpose codes DPM and EGSnrc, respectively, while for photon transport the numbers were 20 and 940. Liu et al. (2012) developed the CPU and GPU-based MC codes for CT organ dose calculation. On a single GPU, the code was found to be 19 times faster than the CPU code and 42 times faster than MCNPX (Pelowitz, 2008). The speedup factors were doubled on a dual-GPU system. Chen et al. (2012)

* Corresponding author.
E-mail address: xug2@rpi.edu (X.G. Xu).

http://dx.doi.org/10.1016/j.anucene.2014.08.061
0306-4549/© 2014 Elsevier Ltd. All rights reserved.

developed an MC tool for CT dose calculations using multi-slice CT (MSCT), flat-detector CT (FDCT), and micro-CT scanners, and observed a speedup factor in the range of 40–50 using a single GPU compared to a single-core CPU.

However, two major problems have not been fully addressed thus far. First, while the achieved speedup factors are impressive in many of the pioneering studies, a fair performance comparison has rarely been emphasized where the CPU is also parallelized to make full use of the available hardware resource. Second, the Intel Xeon Phi MIC coprocessor as a new type of hardware accelerator is a strong competitor of the Nvidia GPU, but its application in MC simulation is still absent in the current literature.

To address these problems, our group have been developing a new parallel MC code called ARCHER (Accelerated Radiation-transport Computation in Heterogeneous EnviRonments) (Xu et al., 2013). The code is designed to accelerate radiation transport simulation in a variety of applications and to make objective evaluation between different platforms. The blueprint of ARCHER is shown in Fig. 1.

In this paper, we describe the development of the photon transport module called ARCHER-CT for fast Monte Carlo CT imaging dose calculation. The module has three components, ARCHER-CT$_{CPU}$, ARCHER-CT$_{GPU}$ and ARCHER-CT$_{COP}$ that are tested on an Intel Xeon X5650 6-core CPU, an Nvidia Tesla M2090 GPU and an Intel Xeon Phi 5110p MIC coprocessor, respectively.

2. Materials and methods

2.1. Physics model

ARCHER-CT simulates the physics of low-energy photons (1–140 keV) in heterogeneous media, where the photoelectric effect, incoherent and coherent scattering are dominant interactions. For the photoelectric effect, the ensuing primary and secondary fluorescence are both explicitly simulated for $Z \geqslant 31$ using the method by Everett and Cashwell (1973). For the incoherent and coherent scattering interactions, the electron's binding effects are being accounted for using the method by Cashwell et al. (1973). Secondary electrons are not simulated, and their energy is assumed to be locally deposited. This is a valid assumption because for the CT application the Continuous Slowing Down Approximation (CSDA) range of electrons inside the phantom is generally one order of magnitude smaller than the dimension of a voxel. The

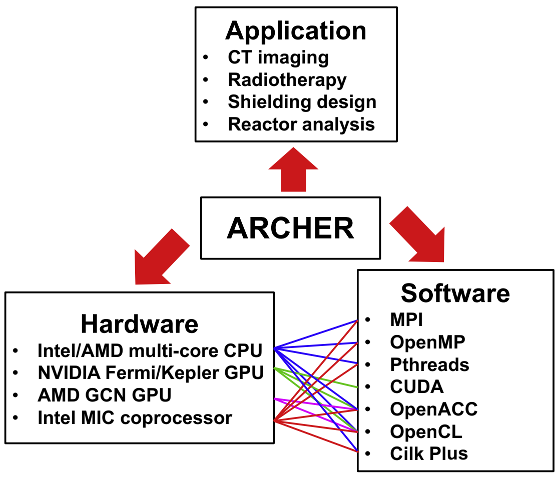

Fig. 1. ARCHER is a versatile research tool for various applications. The code has several components designed for different hardware architectures. The modern hardware architectures usually provide multiple programming models for users to choose from.

capability of electron transport is developed as another module of ARCHER for the radiation therapy application (Su et al., 2014).

The dosimetric quantity of interest in the MC calculation is the absorbed dose D. Under the Transient Charged Particle Equilibrium (TCPE) condition (Attix, 2008), which is usually satisfied in the human body, it is equal to the collision kerma K_c. ARCHER-CT counts D using MCNP's pre-tabulated heating numbers that represent the average energy deposition per collision (X-5 Monte Carlo Team, 2003). The analytical expression of D is given by Eq. (1), where m is the mass of the tallied organ or tissue, E is the photon energy, t is the time, V is the tallied space, $\vec{\Omega}$ is the solid angle, H is the heating number representing the average energy deposition per collision, $\vec{r}$ is the spatial vector, Σ_t is the total linear attenuation coefficient, and ϕ is the angular flux of the photon.

$$D = \frac{1}{m}\int \mathrm{d}E \int \mathrm{d}t \int \mathrm{d}V \int \mathrm{d}\vec{\Omega}\; H(E)\Sigma_t(\vec{r},E)\phi(\vec{r},\vec{\Omega},E,t) \tag{1}$$

In MC method, this quantity is estimated using Eq. (2), where ΔD denotes the dose increment in a single collision event.

$$\Delta D = \frac{H(E)}{m} \tag{2}$$

Special treatment is applied to the calculations of dose to the bone surface and red bone marrow (Schlattl et al., 2007). First, since the bone surface is as thin as 10 μm and cannot be directly modelled in voxel phantoms, its dose is approximated by the dose to the spongiosa (Zhang et al., 2009). Second, the dose to the red bone marrow D_{RBM} is derived from the dose response function $R_{RBM}(E)$, an energy-dependent weighting factor (Zhang et al., 2009), shown in Eq. (3).

$$D_{RBM} = \frac{1}{m}\int \mathrm{d}E \int \mathrm{d}t \int \mathrm{d}V \int \mathrm{d}\vec{\Omega}\; R_{RBM}(E)\phi(\vec{r},\vec{\Omega},E,t) \tag{3}$$

In MC method, this quantity is estimated using the collision estimator in Eq. (4).

$$\Delta D_{RBM} = \frac{R_{RBM}(E)}{m\Sigma_t(\vec{r},E)} \tag{4}$$

2.2. Patient model

ARCHER-CT contains a family of computational human models previously developed by our group, shown in Fig. 2. They include VIP-Man (Xu et al., 2007), RANDO (Wang et al., 2004), RPI-Pregnant women with 3, 6 and 9 months of gestation (Xu et al., 2007), ten extended RPI-Adult females and males representing patients of different Body Mass Indices (BMI), from normal to overweight and to morbidly obese (Zhang et al., 2009; Na et al., 2010; Ding et al., 2012).

2.3. CT scanner model

ARCHER-CT has a built-in GE multi-detector CT (LightSpeed 16 Pro) scanner model developed by Gu et al. (2009), shown in Fig. 3. In this study it is hardcoded into ARCHER-CT. The model has an X-ray point source located at the centre of a sphere. A "cookie cutter" box (Pelowitz, 2008) intersects with the sphere, creating a slot that allows photons to pass and consequently forming a fan beam. The X-ray beam then enters a bowtie filter, whose geometric parameters are determined through an iterative trial-and-error approach to ensure that the simulation result converges to the experimental CTDI values (Gu et al., 2009). The energy spectrum of the source is generated from Xcomp5r (Nowotny and Höfer, 1985). ARCHER-CT allows users to specify the pitch value and choose from a number of predefined parameters, including scan mode (helical, axial),

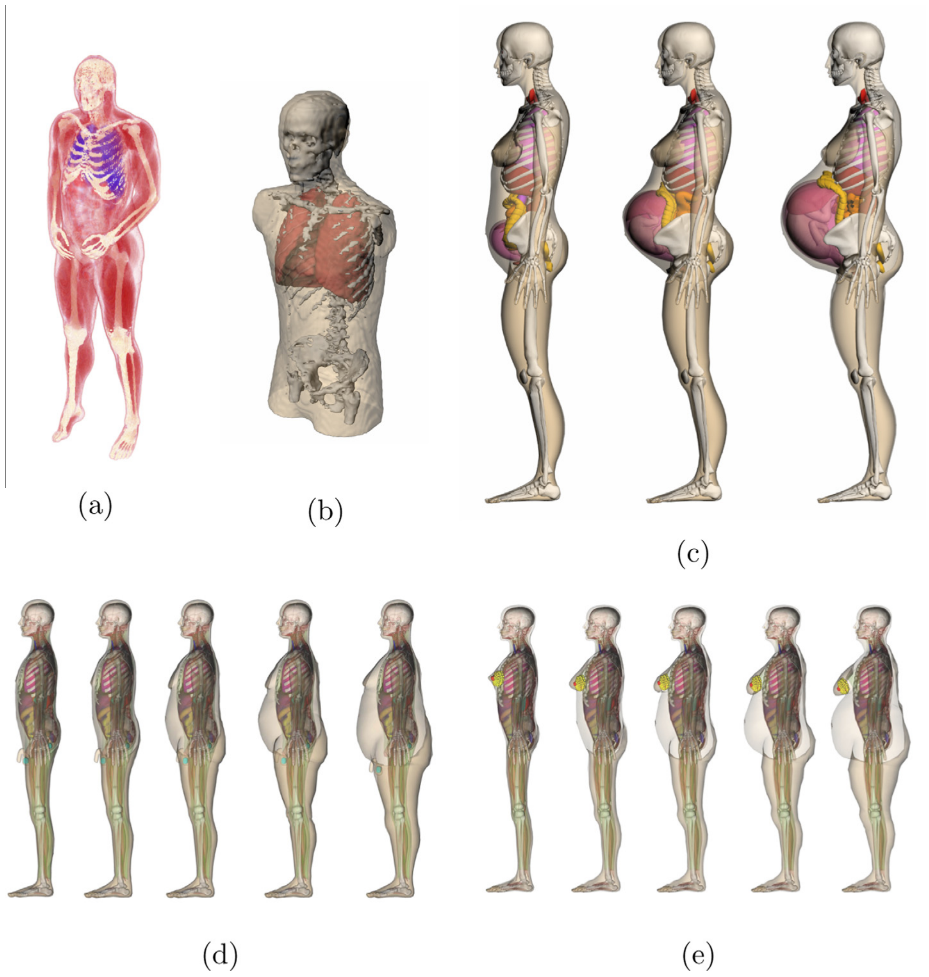

Fig. 2. ARCHER-CT has a built-in phantom library. (a) VIP-Man, (b) RANDO, (c) RPI-Pregnant women with 3, 6 and 9 months of gestation, (d) RPI-Adult females with body weights of 60, 74, 89, 100 and 122 kg, (e) RPI-Adult males with body weights of 73, 86, 103, 117 and 142 kg.

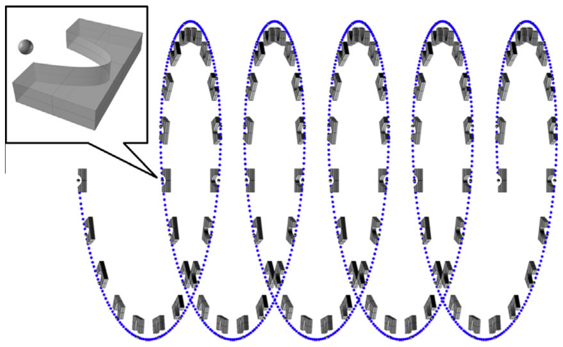

Fig. 3. Trajectory of the CT scanner model in a helical scan. For illustration purpose, a large pitch value is selected.

bowtie filter type (head, body), beam collimation (1.25, 5, 10, or 20 mm) and tube voltage (80, 100, 120 or 140 kVp).

2.4. Hardware architectures

In this study, the heterogeneous computing system is a Tyan FT77B7015 server equipped with an Intel Xeon X5650 CPU (Westmere microarchitecture), an Nvidia Tesla M2090 GPU (Fermi microarchitecture), and an Intel Xeon Phi 5110p coprocessor (Knights Corner microarchitecture). The CPU has 6 cores, and each core supports 2 Intel "hyperthreads" (Intel, 2003). The host system has 16 GB memory.

The GPU is composed of 16 Streaming Multiprocessors (SMs) that share a unified 768 KB L2 cache. Each SM has a bundle of Streaming Processors (SPs) and a local 16/48 KB L1 cache. The parallelism is achieved on two scales: on the global GPU scale, a preset number of GPU threads are grouped into blocks, and the blocks are dispatched by a global thread scheduler to the SMs for independent and simultaneous execution. On the local SM scale, every 32 threads in a block form a warp and execute the same instruction at a time, which is called Single-Instruction, Multiple-Thread (SIMT) model. Instructions from each warp are issued by a local warp scheduler to a group of SPs for pipelined execution. It should be pointed out that, the number of GPU threads is user-specified, but the number of active threads that are resident on the GPU at runtime is always problem-dependent and is limited by the hardware capacity such as the size of registers. Threads will be resident only when hardware resources become available due to the termination of previous resident threads. Programming on Nvidia GPU platform requires an understanding of the interplay between hardware architecture and programming models, and the use of a high-level API called Compute Unified Device Architecture (CUDA) (Nvidia, 2013a). The GPU has 6 GB onboard memory.

The Intel Xeon Phi coprocessor consists of 60 enhanced Pentium-generation cores placed in a ring interconnect, each having private 64 KB L1 and 512 KB L2 caches. The L2 caches are coherent across all cores (Intel, 2013c). Each core supports 4 "hardware threads". There are both similarities and differences between the hardware threads on a coprocessor and the hyperthreads on an Intel CPU. On the one hand, in both implementations threads have their private architectural state such as the registers, while sharing the execution resource such as the execution engine and the cache. On the other hand, the hardware thread on a coprocessor is designed for in-order execution that typically requires 2–4 threads per core for optimal performance, while the hyperthread is for out-of-order execution, whereby it may be beneficial or detrimental to use more than one threads, usually being case-dependent. Programming on Intel Xeon Phi coprocessor requires lower learning curve, because the coprocessor can be considered as a many-core chip not so different from the multi-core CPU, and because it allows a variety of parallel paradigms, including the conventional MPI, OpenMP, POSIX Threads and the new offload pragma (Intel, 2013b) and Cilk Plus (a C language extension for multithreaded parallel computing) (Kim and Voss, 2011). The coprocessor has 8 GB onboard memory.

2.5. Implementation details

The workflow of ARCHER for CT scan simulation is described as follows. (1) The code loads the patient and CT scanner models as well as the photo-atomic interaction database to the memory and performs preprocessing. (2) According to the pre-specified CT scan range, the computation task is divided into a group of independent batches that are treated in a sequential manner. Every batch simulates one scanner rotation where a pre-set number of X-ray photons are tracked in parallel by many threads. The per-batch organ doses and the statistical quantities are calculated and stored. (3) When all the batches have been treated, the overall organ doses and statistical uncertainties are calculated.

The GPU code is written in C using Nvidia's CUDA API. To improve the performance, three issues are carefully considered: the memory usage, execution configuration and concurrency. The GPU provides several types of memory with different features for data storage. The *constant memory* is cached, fast but very small. It is therefore used to store a few physical constants. The *global memory* is slow but large and cached. It is used to hold the large cross-section tables, the phantom and the dose tally data for frequent access. The *texture memory* is a special type of global memory. It has a capability called "hardware filtering" which is to automatically perform linear interpolation in the process of texture fetching (i.e. memory reading) without the need of any code. We experimentally let the texture memory store and interpolate the pre-calculated cumulative distribution function (CDF) tables that are used to sample the polar angle for the incoherent and coherent scattering. The *shared memory* is fast but very small. It is only used to buffer the temporary data coming from the global memory at the end of each batch, where the dose counters from each thread are collected and added up together using Nvidia's reduction algorithm (Harris, 2011).

The execution configuration encompasses the determination of a proper number of threads per block T and the number of blocks per grid B, as well as the proper setup of the GPU hardware. In ARCHER-CT$_{\mathrm{GPU}}$, each thread is assigned a maximum of m photons. The lower limit of m is achieved when the number of threads is maximized such that the total size of dose counters are equal to the GPU memory capacity (6 GB); the upper limit is achieved when the number of threads is minimized such that the GPU is narrowly saturated, i.e. all the computing resource has just been utilized. For the simulation of a batch of $n = 10^7$ photons on the M2090 GPU and 44 dose counters per thread, it can be found that $1 \leqslant m \leqslant 1233$. We set $m = 100$ in our simulations as an appropriate choice. The total number of threads to launch on the GPU t is then n/m, where n is the total number of histories per batch. The number of blocks per grid T is derived from Nvidia occupancy spreadsheet (Nvidia, 2013b). It follows that the number of blocks per grid B is t/T. The GPU hardware is configured through the management tool "nvidia-smi" (Nvidia, 2013c). The "persistent mode" is turned on to reduce the time spent on loading the GPU driver.

The third consideration is the concurrency, referring generally to the ability of a system to perform multiple operations simultaneously (Rennich, 2011). Two types of concurrency are investigated: the single GPU stream concurrency and the CPU-GPU concurrency. The first type of concurrency aims to improve the performance of a single GPU. Because of the random nature of MC simulations, different blocks tend to take different time to complete their jobs. When the simulation of a certain batch is nearing its end, the resident blocks may not suffice to saturate the hardware, leading to a decrease in the GPU occupancy. For a simulation consisting of a sequence of batches, the period of low occupancy can be accumulated to negatively affect the overall GPU performance. This problem can be effectively solved by using the GPU stream. A stream refers to a sequence of commands that execute in order; multiple streams may run concurrently (Nvidia, 2013a). We attach different GPU kernels to separate streams, so that when one kernel on a stream is about to finish and does not fully occupy the hardware resource, kernels on other streams are able to automatically step in and consume the rest of the resource. The second type of concurrency aims to achieve the heterogeneous computing, i.e. making the CPU and GPU work collaboratively. These two computing units adopt an asynchronous execution mode in the sense that once the GPU kernels are launched the control is immediately returned to the CPU, and that the CPU remains idle until the GPU finishes computation (Nvidia, 2013a). To use the untapped multi-core CPU resource, a simplified version of ARCHER-CT$_{\mathrm{CPU}}$ with only the use of multithreading is integrated with the GPU code and executed right after the GPU kernel launch.

The CPU and coprocessor codes are the same, written in C using the MPI/OpenMP model. The three factors described above also apply to the coprocessor code. The problem pertaining to memory usage is simpler, mainly because the coprocessor exposes to users a uniform type of memory to store all the input and output data. To reduce the memory allocation cost, the memory page size is explicitly tuned up from 4 KB to 2 MB using a dedicated "huge page" library (Intel, 2013a). With regard to the execution configuration, we let the coprocessor work in the native execution mode (Intel, 2013c), whereby the executable file, input data and MPI/OpenMP libraries are manually uploaded to the coprocessor, and then the entire code including both the serial and parallel parts are run on it. We issue 1 MPI process and 240 threads on the coprocessor and bind every 4 consecutive threads to the same core. Because the coprocessor does not have the occupancy problem, the task distribution is very straightforward: the photons in a batch are evenly distributed among the all the 240 threads. The concurrency of the CPU and coprocessor is conveniently obtained by using Intel's MPI management tool (Intel, 2014). The CPU-only and coprocessor-only codes are separately compiled, then launched simultaneously by the MPI tool that implicitly takes care of the data transfer.

To ensure a fair performance comparison for the three different hardware architectures the following items are considered. (1) Error-Correcting Code (ECC) is enabled on the GPU and coprocessor. Despite causing a performance reduction of approximately 10%, the ECC functionality improves the hardware reliability when a large amount of jobs are run for a long period. (2) The same pseudo-random number generators Xorshift are used in all the codes (Marsaglia, 2003; Nvidia, 2012). (3) All the codes have approximately the same level of optimization specified by appropriate compiler options. For example, all variants of ARCHER-CT are compiled with a high optimization level `-O3`. Another example is that to improve the performance, some floating point operations are replaced by their faster and less accurate surrogates. The compiler options are `-ffast-math` for the CPU, `-use_fast_math` for the GPU, and `-fp-model fast=2`, `-no-prec-div`, `-no-prec-sqrt`, `-fast-transcendentals` for the coprocessor.

3. Results

3.1. Verification with MCNPX

ARCHER-CT is verified against the production code Monte Carlo N-Particle eXtended (MCNPX) v2.6.0. The whole-body scan over three computational phantoms is simulated respectively, including RPI-Pregnant woman of 9-month gestation, RPI-Adult female obese phantom with 122 kg body weight and RPI-Adult male obese phantom with 142 kg body weight. The CT scan protocol is 20 mm beam collimation, 120 kVp and axial scan mode with a pitch of 1:1. 10^7 photons are simulated per scanner rotation, which restricts the statistical uncertainty of the overall doses to 0.5%. The absorbed doses to organs/tissues of dosimetric interest listed in ICRP 103 recommendations are calculated, and compared to the results of MCNPX. The percentage difference between ARCHER-CT$_{GPU}$ and MCNPX defined as $(\text{ARCHER} - \text{CT}_{\text{GPU}} - \text{MCNPX})/\text{MCNPX} \times 100\%$ is shown in Figs. 4–6. The absolute percentage difference across these three cases is 0.38% on average, indicating an excellent agreement. Besides, the difference between the results by ARCHER-CT$_{GPU}$ and those by ARCHER-CT$_{CPU}$ or ARCHER-CT$_{COP}$ is found to be negligible. The difference itself is primarily caused by the unique number of threads and the number of photons simulated per thread (hence the unique number of pseudo-random number streams and the amount of random numbers consumed by each stream) used for each code.

3.2. Comparison of computing efficiency

The parallel execution is configured as follows. The parallel MCNPX uses 12 MPI processes on the Intel Xeon X5650 6-core CPU. ARCHER-CT$_{CPU}$ uses 1 MPI process and 12 threads on the same CPU. ARCHER-CT$_{GPU}$ uses 10^5 light-weight GPU threads on the Nvidia M2090 GPU for each batch, and the actual number of resident threads at runtime is approximately 8000. ARCHER-CT$_{COP}$ uses 1 MPI processes and 240 threads on the Intel Xeon Phi 5110p coprocessor.

The computation time by different codes using different phantoms is listed in Tables 1–3. All the ARCHER-CT variants are computationally efficient and are substantially faster than the parallel MCNPX. It is important to mention that the remarkable performance difference has three causes: (1) MCNPX used in this study is a precompiled executable with `-O1` optimization level and double precision floating point (FP64) calculation, while ARCHER-CT applies more aggressive `-O3` optimizations and single precision (FP32) calculation. (2) There are several major algorithm differences. First, ARCHER-CT uses an improved method for biased source sampling, in which the initial photon position is bounded by the slot created by the "cookie cutter" object mentioned in Section 2.3, leading to a higher acceptance rate in the rejection sampling process. Second, ARCHER-CT uses Woodcock delta tracking method

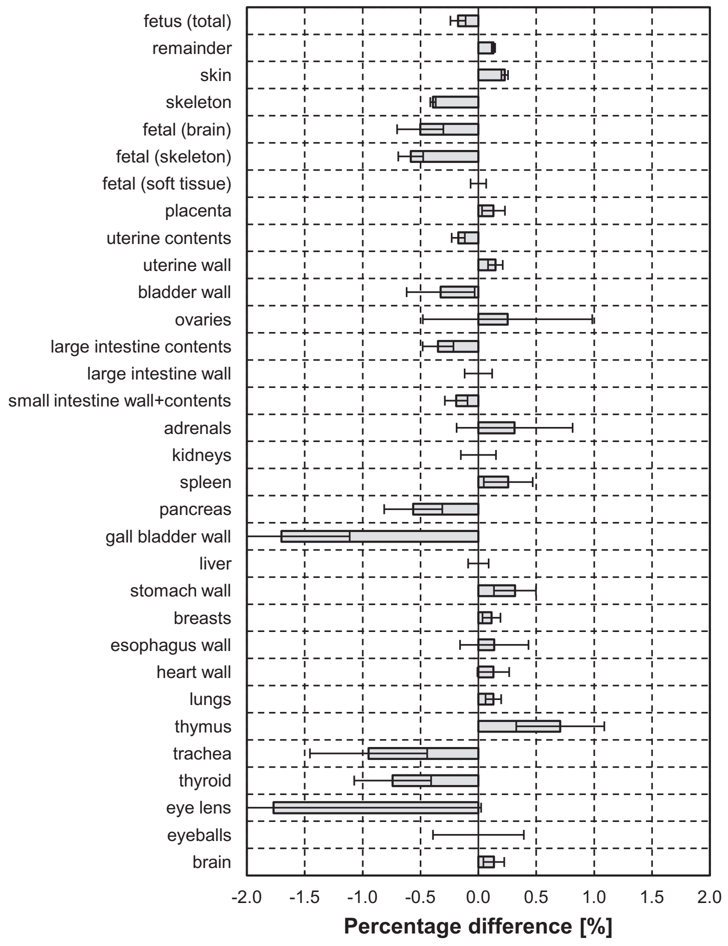

Fig. 4. Percentage difference between ARCHER-CT and MCNPX using RPI-Pregnant women (9-month gestation) phantom. A total of 8.2×10^8 photons are simulated.

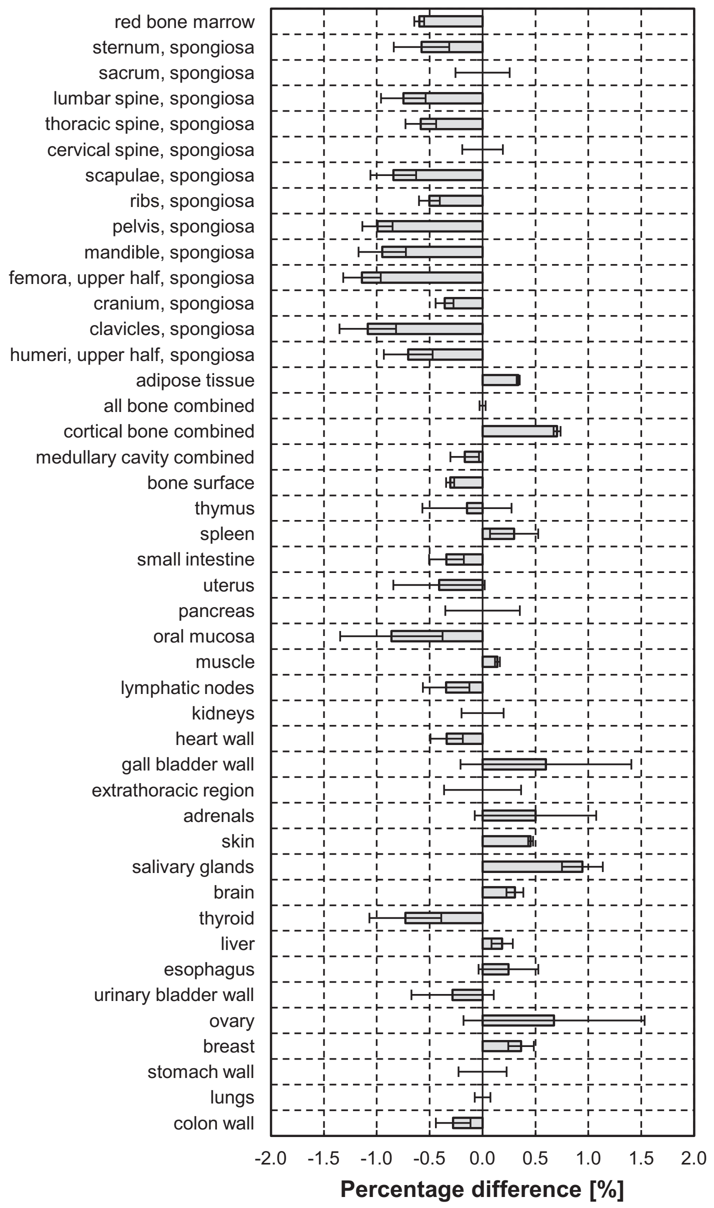

Fig. 5. Percentage difference between ARCHER-CT and MCNPX using RPI-Adult female obese phantom (122 kg). A total of 8.3×10^8 photons are simulated.

to sample the path-length, which is a more efficient alternative to the conventional surface-to-surface ray-tracing in MCNPX. Third, to accelerate the simulation process, ARCHER-CT obtains the polar angle of the scattering interaction by table lookup and interpolation instead of the on-the-fly rejection sampling used in MCNPX. (3) MCNPX is a general-purpose production MC code that supports a wide spectrum of applications beyond medical physics alone. In contrast, ARCHER-CT is a specialized code developed specifically for CT simulations, and has been optimized for that specific class of computational models.

Because of the above reasons, we use ARCHER$_{CPU}$ as the baseline to derive the speedup factors when comparing different computing platforms. While both are much faster than the CPU baseline, the GPU code is found to outperform the coprocessor code in all the three test cases by 55–72%. Besides, the heterogeneous computing mode where the GPU and CPU work concurrently produces a performance gain of 13–15% compared to the case where the GPU works alone. Likewise, the collaboration between coprocessor and CPU makes the simulation faster by 18% than a single coprocessor.

4. Discussion

4.1. Atomic summation versus parallel summation

The GPU code launches thousands of active threads at runtime. There are two typical ways to register the photon energy deposition from each thread. One is the atomic summation, which serializes the addition operations from different threads but requires less memory space. The other is the parallel summation, which has each thread independently keep their individual copy of the dose data until the global reduction and is more memory intensive. For the former method, however, as the problem size scales up, the numerical error will become increasingly noticeable, as illustrated by Fig. 7. In this test case, dose to the lung in a chest region CT scan is calculated. The atomic summation fails after the number of photons exceeds 10^8, the result being smaller than the true value. This deviation is due to the fact that as the sum becomes larger, more and more low-order digits of a small floating point number added to it are discarded. In contrast, the result obtained by parallel summation maintains good consistency, being 0.8% different from

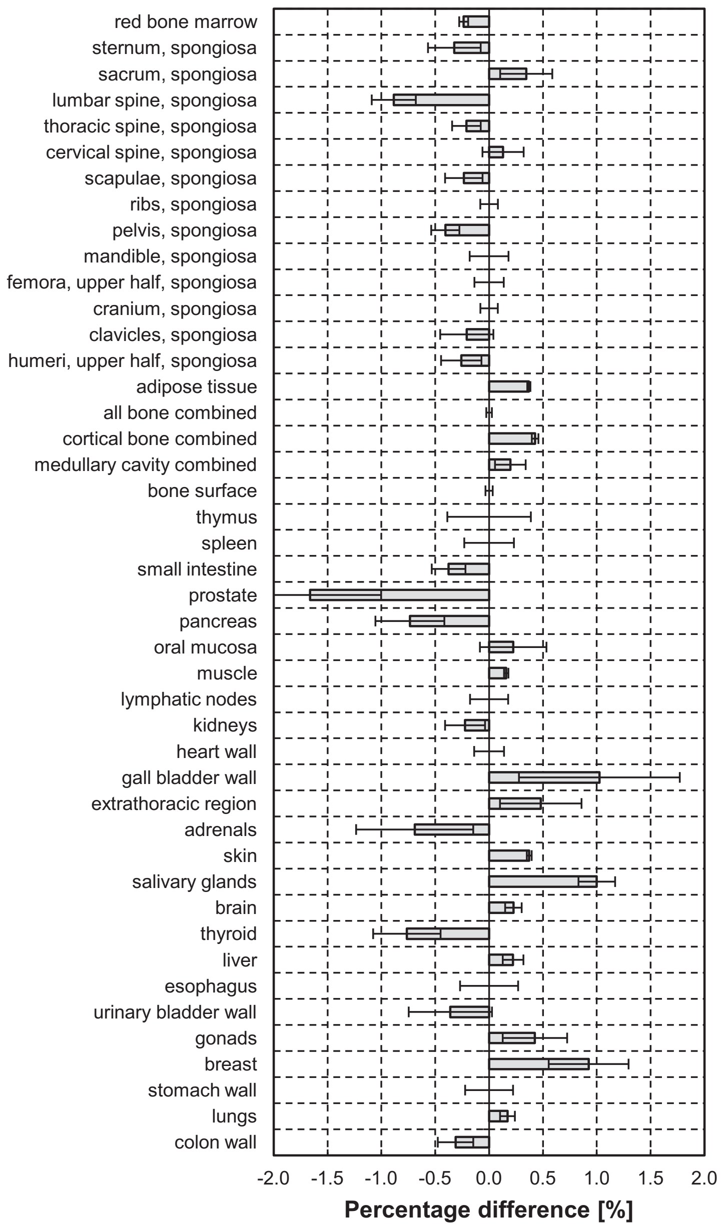

Fig. 6. Percentage difference between ARCHER-CT and MCNPX using RPI-Adult male obese phantom (142 kg). A total of 9×10^8 photons are simulated.

Table 1
Computation time by MCNPX, ARCHER$_{CPU}$, ARCHER$_{GPU}$, and ARCHER$_{COP}$ using RPI-Pregnant women (9-month gestation) phantom.

Code	Execution time [min]	Speedup
parallel MCNPX (12 MPI processes)	479.89	–
parallel ARCHER$_{CPU}$	8.03	Baseline
ARCHER$_{GPU}$	1.38	5.81×
ARCHER$_{GPU+CPU}$	1.23	6.54×
ARCHER$_{COP}$	2.38	3.38×
ARCHER$_{COP+CPU}$	2.02	3.98×

Table 2
Computation time by MCNPX, ARCHER$_{CPU}$, ARCHER$_{GPU}$, and ARCHER$_{COP}$ using RPI-Adult female obese phantom (122 kg).

Code	Execution time [min]	Speedup
Parallel MCNPX (12 MPI processes)	460.28	–
Parallel ARCHER$_{CPU}$	10.59	Baseline
ARCHER$_{GPU}$	2.06	5.15×
ARCHER$_{GPU+CPU}$	1.81	5.85×
ARCHER$_{COP}$	3.21	3.30×
ARCHER$_{COP+CPU}$	2.72	3.89×

MCNPX result. In this method, which is based on the classic pairwise summation, the two floating pointer numbers added together are generally not several orders of magnitude different; hence a smaller numerical error. Because of this consideration, in ARCHER-CT each thread holds its individual dose counters, and the parallel summation technique (implemented through Nvidia's fast algorithm (Harris, 2011)) is used to gather the results from a large number of counters for the organ dose calculation.

Sometimes the atomic summation may appear inevitable. For example, when a 3-D dose distribution is desired, it is not feasible to have each thread hold a large tally matrix with the same size of a voxelized phantom, due to the limited amount of memory available. The numerical errors introduced by the atomic operation can be circumvented by using the double precision floating point arithmetic. Although currently the double precision atomic addition is not directly supported by the CUDA GPUs, a compare-and-swap

Table 3
Computation time by MCNPX, ARCHER$_{CPU}$, ARCHER$_{GPU}$, and ARCHER$_{COP}$ using RPI-Adult male obese phantom (142 kg).

Code	Execution time [min]	Speedup
Parallel MCNPX (12 MPI processes)	1421.73	–
Parallel ARCHER$_{CPU}$	11.48	Baseline
ARCHER$_{GPU}$	2.23	5.15×
ARCHER$_{GPU+CPU}$	1.93	5.94×
ARCHER$_{COP}$	3.44	3.33×
ARCHER$_{COP+CPU}$	2.92	3.94×

algorithm devised by Nvidia can be used as an effective emulation method (Nvidia, 2013a). The disadvantage is that this method can greatly increase the computation time.

4.2. Performance bottleneck

Despite the good performance of the GPU and coprocessor code observed in this study, it should be pointed out that the capability of floating point calculation on these two hardware accelerators is still underutilized.

For the Nvidia GPU, there are two major limiting factors, the branch divergence and the scattered memory access pattern, both stemming from the statistical nature of MC method. The GPU hardware has an elegant mechanism to handle the divergent branches while maintaining the SIMT execution model. For instance, when encountering an if-else branch, the GPU picks up one of the branch to execute. The threads in a warp that are not supposed to fall into that branch will be temporarily masked out (i.e. their operation has no effect on the memory), until that branch is finished and the other branch is started where the threads are reactivated. Code with frequent occurrence of divergence typically leads to many disabled threads. To analyse ARCHER$_{GPU}$ quantitatively, we use Nvidia's tool "nvprof" (Nvidia, 2013d) to profile the code. The data are shown in Table 4. The high branch efficiency means that for our calculation, quite counter-intuitively, the threads within a GPU warp tend to uniformly follow the same branch for most of the cases. However, the low warp execution efficiency indicates that those rare threads that diverge from others tend to execute a large amount of instructions, during which period their peers are masked out and a waste of the GPU resource results.

Furthermore, the performance of the GPU code is usually associated with the memory access pattern. Ideally, consecutive threads in a warp ask data from consecutive locations in the global memory space that are within one 128-Byte segment. Under this circumstance, the warp only takes one memory transaction to fetch 32 single-precision floating point data for its 32 threads, and the global load transactions per warp request is 1. In ARCHER$_{GPU}$, access to the cross-section data is highly irregular, and repeated, expensive memory transaction has to be conducted to feed all the threads in a warp, causing a performance downgrade.

For the Intel Xeon Phi coprocessor, the limiting factor is that the distinctive vectorization feature has not yet been efficiently used. Specifically, each core of the coprocessor has a vector processing unit (VPU) with 512-bit wide registers for the single instruction multiple data (SIMD) operations (Intel, 2013c) However, the conventional history-based MC algorithm currently adopted in ARCHER$_{COP}$ makes it difficult to directly benefit from that feature. There is very limited room for vectorization. One way is to rely on the compiler-driven automatic vectorization. The other is the

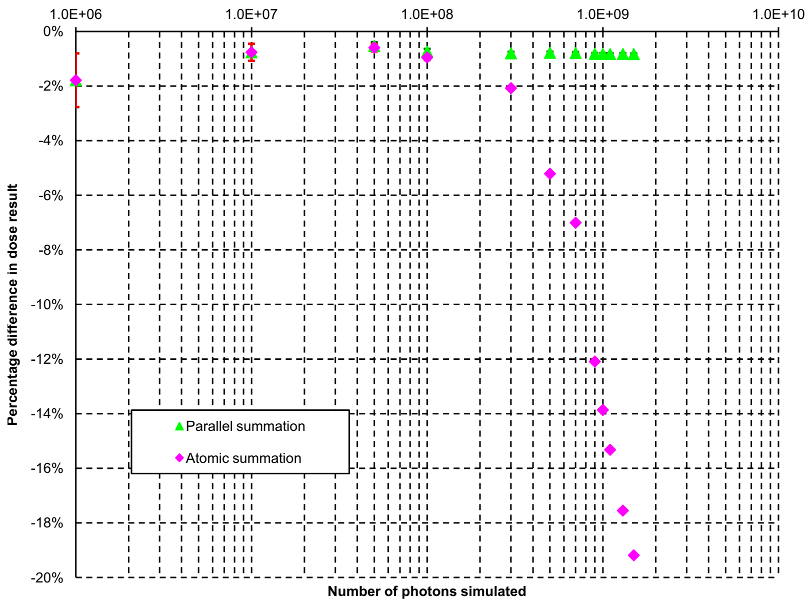

Fig. 7. The influence of parallel and atomic summation methods over the accuracy of the lung dose in a simulated chest scan.

Table 4
Profiling result of ARCHER$_{GPU}$. The simulation is a single axial CT scan over the abdominal region of the RPI-Adult male obese phantom (142 kg) using 10^7 photons.

Metric	Meaning	Value
Branch efficiency	Ratio of non-divergent branches to total branches	92.35%
Warp execution efficiency	Ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor	15.69%
Global load transactions per warp request	Average number of global memory load transactions performed for each global memory load	2.6

manual vectorization by adding the compiler directives, such as `#pragma ivdep` to the loop structures which should not have data dependencies. Both methods only apply to a limited number of inner for-loops in the MC transport kernel and do not lead to appreciable performance improvement. A potential future direction is to implement the vectorized MC algorithm (Brown and Martin, 1984) on the coprocessor.

5. Conclusions

In this paper, we presented the development of a new Monte Carlo-based code ARCHER-CT for fast CT imaging organ dose calculations on heterogeneous computing systems. The code has three components, $\text{ARCHER}_{\text{CPU}}$, $\text{ARCHER}_{\text{GPU}}$, and $\text{ARCHER}_{\text{COP}}$ that are designed to be run on the multi-core CPU, Nvidia GPU and Intel Xeon Phi coprocessor, respectively. The latter two are collectively called hardware accelerators.

ARCHER-CT contains a validated GE multi-detector CT scanner (LightSpeed 16 Pro) model and a library of computational human phantoms such as VIP-Man, RANDO, RPI-Pregnant women with 3, 6 and 9 months of gestation, ten extended RPI-Adult females and males representing patients of different BMIs.

To make a reasonable, objective comparison, much effort has been made to optimize the three codes to maximize the performance on their specific architectures. $\text{ARCHER}_{\text{CPU}}$ written in MPI-OpenMP hybrid is tested using 12 threads; $\text{ARCHER}_{\text{GPU}}$ written in CUDA C launches approximately 8000 active threads at runtime; $\text{ARCHER}_{\text{COP}}$ written in MPI-OpenMP uses 240 threads. Furthermore, the CPU has the hyperthreading function enabled to increase the performance, and the hardware accelerators are configured to work in the server mode with ECC enabled. The same random number generator and similar compiler options are used in the three codes.

In the accuracy and performance tests, three inhomogeneous phantoms are used: RPI-Pregnant woman (9-month), RPI-Adult female obese phantom (122 kg) and RPI-Adult male obese phantom (142 kg). Absorbed doses to individual organs/tissues are calculated and are found to be in excellent agreement with the production code MCNPX. While significantly faster than the parallel MCNPX run with 12 MPI processes, it is observed that the performance of $\text{ARCHER}_{\text{GPU}}$ and $\text{ARCHER}_{\text{COP}}$ are 5.15–5.81 and 3.30–3.38 times faster than the parallel $\text{ARCHER-CT}_{\text{CPU}}$. The GPU code outperforms the coprocessor code in all three test cases. It is beneficial to make CPU work concurrently with the GPU and coprocessor, and the performance gain is found to be 13–18%.

Acknowledgments

This project is supported by the National Institute of Biomedical Imaging and Bioengineering through an R01 grant (R01EB015478). Former Rensselaer Radiation Measurement and Dosimetry Group (RRMDG) members, Dr. Aiping Ding and Dr. Matthew Mille, participated in GPU software development at RPI during 2008–2011. We would like to thank Dr. Mark S. Shephard from RPI and Dr. Forrest B. Brown from Los Alamos National Laboratory (LANL) for their invaluable technical support, and Nvidia for the generous hardware donation.

References

Attix, F.H., 2008. Introduction to Radiological Physics and Radiation Dosimetry. John Wiley & Sons, Weinheim, Germany.
Badal, A., Badano, A., 2009. Accelerating Monte Carlo simulations of photon transport in a voxelized geometry using a massively parallel graphics processing unit. Med. Phys. 36 (11), 4878–4880.
Badal, A., Badano, A., 2011. Fast and accurate estimation of organ doses in medical imaging using a GPU-accelerated monte carlo simulation. Med. Phys. 38 (6), 3411.
Baró, J., Sempau, J., Fernández-Varea, J., Salvat, F., 1995. Penelope: An algorithm for Monte Carlo simulation of the penetration and energy loss of electrons and positrons in matter. Nucl. Instrum. Meth. B 100 (1), 31–46.
Brown, F.B., Martin, W.R., 1984. Monte Carlo methods for radiation transport analysis on vector computers. Prog. Nucl. Energ. 14 (3), 269–299.
Cashwell, E.D., Neergaard, J.R., Everett, C.J., Schrandt, R.G., Taylor, W.M., Turner, G.D., 1973. Monte Carlo photon codes: MCG and MCP. Report, Los Alamos National Lab (LANL).
Chen, W., Kolditz, D., Beister, M., Bohle, R., Kalender, W.A., 2012. Fast on-site Monte Carlo tool for dose calculations in CT applications. Med. Phys. 39 (6), 2985–2996.
Ding, A., Mille, M.M., Liu, T., Caracappa, P.F., G, X.X., 2012. Extension of RPI-adult male and female computational phantoms to obese patients and a Monte Carlo study of the effect on ct imaging dose. Phys. Med. Biol. 57 (9), 2441–2459.
Everett, C.J., Cashwell, E.D., 1973. MCP code fluorescence-routine revision. Report LA-5240-MS, Los Alamos National Laboratory (LANL).
Gu, J., Bednarz, B., Caracappa, P.F., Xu, X.G., 2009. The development, validation and application of a multi-detector CT (MDCT) scanner model for assessing organ doses to the pregnant patient and the fetus using Monte Carlo simulations. Phys. Med. Biol. 54 (9), 2699–2717.
Harris, M., 2011. Optimizing parallel reduction in CUDA, http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf (Retrieved on July 09, 2014).
Hissoiny, S., Ozell, B., Bouchard, H., Després, P., 2011. GPUMCD: a new GPU-oriented Monte Carlo dose calculation platform. Med. Phys. 38 (2), 754–764.
Intel, 2003. Intel Hyper-Threading technology technical user's guide. Available from: http://cache-www.intel.com/cd/00/00/01/77/17705_htt_user_guide.pdf (Retrieved on July 09, 2014).
Intel, 2013a. How to use huge pages to improve application performance on Intel Xeon Phi coprocessor. Available from: https://software.intel.com/sites/default/files/Large_pages_mic_0.pdf (Retrieved on July 09, 2014).
Intel, 2013b. Intel C++ compiler XE 13.1 user and reference guide. Available from: https://software.intel.com/sites/products/documentation/doclib/stdxe/2013/composerxe/compiler/cpp-win/index.htm (Retrieved on July 09, 2014).
Intel, 06/2013 2013c. Intel Xeon Phi coprocessor system software developers guide. Available from: https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system-software-developers-guide (Retrieved on July 09, 2014).
Intel, 2014. Intel MPI library for Linux OS reference manual. Available from: https://prd1idz.cps.intel.com/sites/products/documentation/hpc/ics/impi/41/lin/Reference_Manual/Reference_Manual.pdf (Retrieved on July 09, 2014).
Jia, X., Gu, X., Graves, Y.J., Folkerts, M., Jiang, S.B., 2011. GPU-based fast Monte Carlo simulation for radiotherapy dose calculation. Phys. Med. Biol. 56 (22), 7017–7031.
Jia, X., Gu, X., Sempau, J., Choi, D., Majumdar, A., Jiang, S.B., 2010. Development of a GPU-based Monte Carlo dose calculation code for coupled electronphoton transport. Phys. Med. Biol. 55 (11), 3077–3086.
Keckler, S.W., Dally, W.J., Khailany, B., Garland, M., Glasco, D., 2011. GPUs and the future of parallel computing. Micro. IEEE 31 (5), 7–17.
Kim, W., Voss, M., 2011. Multicore desktop programming with intel threading building blocks. Software IEEE 28 (1), 23–31.
Liu, T., Ding, A., Xu, X.G., 2012. Accelerated Monte Carlo methods for photon dosimetry using a dual-GPU system and cuda. Med. Phys. 39 (6), 3818.
Marsaglia, G., 2003. Xorshift RNGs. J. Stat. Software 8 (14), 1–6.
Na, Y.H., Zhang, B., Zhang, J., Caracappa, P.F., Xu, X.G., 2010. Deformable adult human phantoms for radiation protection dosimetry: anthropometric data representing size distributions of adult worker populations and software algorithms. Phys. Med. Biol. 55 (13), 3789–3811.
Nowotny, R., Höfer, A., 1985. Ein Programm für die Berechnung von diagnostischen Röntgenspektren. Fortschr Röntgenstr 142 (6), 685–689.
Nvidia, 2012. CUDA toolkit 4.2 curand guide.
Nvidia, 2013a. CUDA C programming guide. Available from: http://docs.nvidia.com/cuda/cuda-c-programming-guide/ (Retrieved on July 09, 2014).
Nvidia, 2013b. GPU occupancy calculator. Available from: http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls (Retrieved on July 09, 2014).
Nvidia., 2013a. NVIDIA System Management Interface program. Available from: http://developer.download.nvidia.com/compute/cuda/5_5/rel/nvml/nvidia-smi.5.319.43.pdf (Retrieved on July 09, 2014).
Nvidia., 2013b. Profiler user's guide. Available from: http://docs.nvidia.com/cuda/pdf/CUDA_Profiler_Users_Guide.pdf (Retrieved on July 09, 2014).
Pelowitz, D.B., 2008. MCNPX user's manual, version 2.6.0. Report LA-CP-07-1473, Los Alamos National Laboratory (LANL).
Rennich, S., 2011. CUDA C/C++ streams and concurrency. Available from: http://on-demand.gputechconf.com/gtc-express/2011/presentations/StreamsAndConcurrencyWebinar.pdf (Retrieved on July 09, 2014).
Schlattl, H., Zankl, M., Petoussi-Henss, N., 2007. Organ dose conversion coefficients for voxel models of the reference male and female from idealized photon exposures. Phys. Med. Biol. 52 (8), 2123–2145.
Sempau, J., Wilderman, S.J., 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 (8), 2263–2291.
Su, L., Yang, Y., Bednarz, B., Sterpin, E., Du, X., Liu, T., Ji, W., Xu, X.G., 2014. $\text{ARCHER}_{\text{RT}}$, A photon-electron coupled Monte Carlo dose computing engine for GPU: software development of and application to helical tomotherapy. Med. Phys. 41 (7), 071709.
Top500, 2014. Top 500 supercomputer sites. Available from: http://www.top500.org/lists/ (Retrieved on July 09, 2014).

Wang, B., Xu, X.G., Kim, C.H., 2004. A Monte Carlo CT model of the rando phantom. Trans. Am. Nucl. Soc. 90, 473–474.

X-5 Monte Carlo Team, 04 2003. MCNP-a general Monte Carlo N-Particle Transport code, Version 5. Volume I: Overview and theory. Report LA-UR-03-1987, Los Alamos National Laboratory (LANL).

Xu, X.G., Liu, T., Su, L., Du, X., Riblett, M., Ji, W., Brown, F.B., 2013. An update of ARCHER, a Monte Carlo radiation transport software testbed for emerging hardware such as GPUs. Trans. Am. Nucl. Soc. 108, 433–434.

Xu, X.G., V, T., J, Z., C, S., 2007. A boundary-representation method for designing whole-body radiation dosimetry models: pregnant females at the ends of three gestational periods-RPI-p3, -p6 and -p9. Phys. Med. Biol. 52 (23), 7023–7044.

Zhang, J., Na, Y.H., Caracappa, P.F., G, X.X., 2009. RPI-AM and RPI-AF, a pair of mesh-based, size-adjustable adult male and female computational phantoms using ICRP-89 parameters and their calculations for organ doses from monoenergetic photon beams. Phys. Med. Biol. 54 (19), 5885–5908.