Annals of Nuclear Energy 82 (2015) 2–9

Contents lists available at ScienceDirect

Annals of Nuclear Energy

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

ARCHER, a new Monte Carlo software tool for emerging heterogeneous computing environments

X. George Xu [a,*], Tianyu Liu [a], Lin Su [a], Xining Du [a], Matthew Riblett [a], Wei Ji [a], Deyang Gu [a], Christopher D. Carothers [a], Mark S. Shephard [a], Forrest B. Brown [b], Mannudeep K. Kalra [c], Bob Liu [c]

[a] Rensselaer Polytechnic Institute, Troy, NY, USA
[b] Los Alamos National Laboratory, Los Alamos, NM, USA
[c] Massachusetts General Hospital, Boston, MA, USA

ARTICLE INFO

Article history:
Received 23 April 2014
Accepted 27 August 2014
Available online 28 October 2014

Keywords:
Monte Carlo
Parallel computing
GPU
CUDA
Intel Xeon Phi coprocessor
Radiation dose

ABSTRACT

The Monte Carlo radiation transport community faces a number of challenges associated with peta- and exa-scale computing systems that rely increasingly on heterogeneous architectures involving hardware accelerators such as GPUs and Xeon Phi coprocessors. Existing Monte Carlo codes and methods must be strategically upgraded to meet emerging hardware and software needs. In this paper, we describe the development of a software, called ARCHER (Accelerated Radiation-transport Computations in Heterogeneous EnviRonments), which is designed as a versatile testbed for future Monte Carlo codes. Preliminary results from five projects in nuclear engineering and medical physics are presented.

1. Introduction

Hardware accelerators that emerged in recent years have become increasingly popular for high performance computing (HPC). Such hardware facilities include Nvidia and AMD's graphics computing units (GPUs) and Intel's many-integrated-core (MIC) architecture based coprocessors. With high energy efficiency and strong computing power, these accelerators are being used in HPC facilities including both workstations and supercomputers. The accelerators work together with the traditional central processing units (CPUs) and give the system additional performance boosts. Systems consisting of both CPUs and hardware accelerators belong to the so-called "heterogeneous architectures".

As of June 2013, four out of the top ten supercomputers on the Top500 list (TOP500, 2013) use the heterogeneous design involving either GPUs or coprocessors. Tianhe-2, the No. 1 supercomputer, has 32,000 Intel Xeon E5-2692 12-core CPUs and 48,000 Intel Xeon Phi 31S1P coprocessors to claim a peak performance of 54,902 Tflops. Fallen behind is the No. 2 supercomputer, Titan, which uses 560,640 AMD Opteron 6274 16-core CPUs and 261,632 NVIDIA K20x GPUs to reach its peak performance of 27,112 Tflops.

It is likely that the exa-scale computing era, which will arrive by the end of this decade, must utilize a drastically new architecture. However, none of the production Monte Carlo (MC) radiation transport codes widely used by the nuclear engineering and radiology communities was designed to take advantage of today's heterogeneous computers. One challenge has to do with the sometimes prohibitively large amount of time required to port an existing MC code to the new hardware platform. Given the uncertainty associated with the final hardware/software specifications of the exa-scale supercomputer systems, most software developers and end-users are reluctant to play an active role in the "co-design" process.

Only a handful groups have managed to make breakthroughs in designing new, GPU-based MC codes. For example, (Badal and Badano, 2009) developed MC-GPU for radiographic projection and CT dose calculations, and reported a speedup factor of 110 over general-purpose MC code PENELOPE (Baro et al., 1995). Jia et al. (2011) developed gDPM based on the dose planning method (DPM) (Sempau et al., 2000) and found a speedup factor of 69.1–87.2 over the original DPM package on CPU. Hissoiny et al. (2011) developed GPUMCD for coupled electron-photon transport and reported a speedup factor of 1200 for electron beams and a speedup factor of 940 for photon beams when compared with a production code, EGSnrc. Ding et al. (2011) and Liu et al. (2012b,c) reported the GPU-based MC code for X-ray CT dosimetry using a validated CT model. The initial test code was found to be

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

http://dx.doi.org/10.1016/j.anucene.2014.08.062
0306-4549/

19 times faster than the corresponding CPU code and 42 times faster than the general-purpose MC code MCNPX (Pelowitz, 2011). Jahnke et al. (2012) developed GMC based on the general purpose MC code, Geant4 (Carrier et al., 2004), and observed a speedup factor of 4860 for the GPU code compared to Geant4. Recently Chen et al. (2012) implemented a fast MC tool for dose calculations and reported that the GPU code is 55 times faster than the single-thread CPU code for a thorax phantom CT scan.

GPU-based MC methods have also been applied to other types of particles in nuclear and medical applications. Nelson and Ivanov (2010) studied the neutron transport problem on GPUs and reported a speedup factor of 23 with single precision floating point. Heimlich et al. (2011) calculated the penetration probability of a neutron beam incident on a homogeneous 1D slab, and reported a speedup factor of 125 times to a single-core CPU. Liu et al. (2012a) studied the neutron eigenvalue problem for a bare sphere core and a binary slab system, and reported a speedup factor of 7 and 33.3 for two geometry configurations, respectively. Other groups have developed the GPU-based MC code for proton calculations. Yepes et al. (2010) implemented a GPU code GFDC using the track-length algorithm for proton dose calculations and observed a speedup factor of 75 with respect to the CPU-based code FDC. Su et al. (2012) studied the proton beam depth dose in a 3D water phantom and reported the GPU code is 57 times faster than the CPU implementation. Jia et al. (2012) developed gPMC for proton dose calculations in radiotherapy and found the typical computation time ranges from 6 s to 22 s instead of several hours using CPU-based code.

The effort made by the MC community, however, is still at an early stage. Several problems have been found in most existing studies: relatively simple physics models were often used and the effects of using more realistic physics on GPU speedup factors were not carefully investigated. Furthermore, different physics models and optimization levels were often used for CPUs and GPUs, resulting in unfair performance comparisons. Finally, the Intel Xeon Phi coprocessors, which became available recently as a hardware accelerator comparable to GPUs, has not been adopted for MC based radiation transport simulations.

In this paper, we describe the development of a software testbed, called ARCHER (Accelerated Radiation-transport Computations in Heterogeneous EnviRonments). The paper first discusses basic design features, followed by examples of using ARCHER for various test cases. Fig. 1 illustrates the long-term vision of ARCHER as a comprehensive Monte Carlo software testbed, using the novel computing hardware and advanced programming models to speed up Monte Carlo calculations. Currently we employ ARCHER as a versatile research tool to evaluate the performance of Nvidia's GPU and Intel's coprocessor. Eventually, ARCHER can evolve into a suite of MC codes with the possibility of being compatible with most existing and next-generation exa-scale supercomputers.

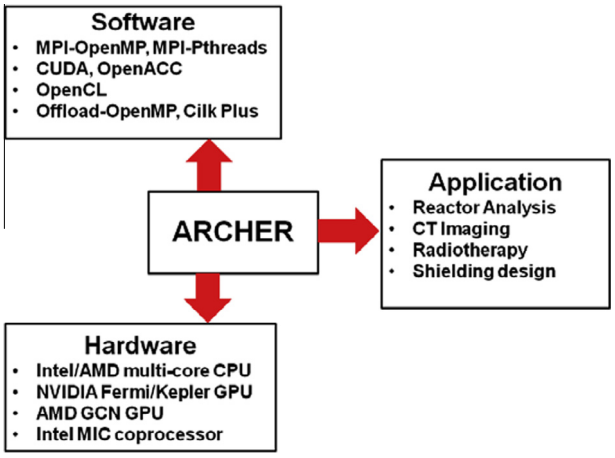

Fig. 1. The vision of ARCHER as a research tool to study MC simulations on a variety of hardware platforms including CPUs, GPUs and coprocessors as well as software tools including MPI, OpenMP, CUDA for diverse applications ranging from nuclear reactor engineering to medical/health physics.

2. Methods

As an MC-based code, ARCHER currently can simulate the transport of photon, electron and neutrons in 3D heterogeneous media.

2.1. Photon transport

ARCHER simulates Rayleigh scattering, photoelectric effect, Compton scattering, and pair production for photons ranging from 1 keV to 20 MeV.

For performance comparison, three physics models with different complexity levels were developed: (1) Detailed physics, where atomic form factors are used to account for electron's binding effects over energy and angular distribution, and the attenuation coefficients are calculated from raw microscopic cross sections on the fly. (2) Simple physics, where atomic form factors are ignored and pre-tabulated cross section data are used for efficiency. (3) Combo physics, where the material attenuation coefficients are calculated by using pre-tabulated cross section data. The difference from the simple physics mode is that the polar angles in the coherent and incoherent scattering interactions are sampled directly by using pre-calculated tables that take into account the effects of form factors.

Photon transport simulations are used in a variety of applications including CT dose calculations and radiotherapy treatment. A dedicated module called ARCHER-CT is developed to perform fast and accurate dose calculation for CT scan simulations. In ARCHER-CT, only low energy ($E \leqslant 140$ keV) photons are transported in computational human phantoms while secondary electrons are assumed to deposit their energy locally. This is valid because the Continuous Slowing Down Approximation (CSDA) range for electrons in the energy range considered is generally one order of magnitude smaller than the dimension of a phantom voxel.

A family of voxelized computational human phantoms were incorporated in ARCHER-CT, including RPI Pregnant Women and RPI-Adult Males and Adult-Females with different body weights, as shown in Fig. 2 Ding et al., 2012.

The CT scanner model used in ARCHER-CT was originally defined by Gu et al. (2009) in MCNP input data format for a 3rd-generation GE LightSpeed 16 multi-detector CT scanner. The scanner model, which includes a curved X-ray source surface and a bowtie filter, was then hardcoded into ARCHER-CT. The code supports several predefined scan protocols including helical or axial scan modes, beam collimations of 1.25, 5, 10, or 20 mm and kVps of 80, 100, 120 or 140.

Organ doses are tallied using the collision estimator. In general there are two methods to update the estimator. The first is called the atomic operation in which multiple threads attempting to increment the estimator located in one memory location are serialized to ensure a correct result. We used the second method, in which each thread keeps its own local dose array and the final dose results are obtained by summing over the corresponding local dose array elements. This approach effectively avoids the loss of accuracy that has been observed in the first method when a large number of photons ($\sim 10^8$) are simulated and a small number of organs ($\sim$10–100) are tallied (Liu et al., 2013).

For performance comparison, a CPU-based code, $\text{ARCHER}_{\text{CPU}}$, is first developed as a basis. The shared memory model OpenMP is

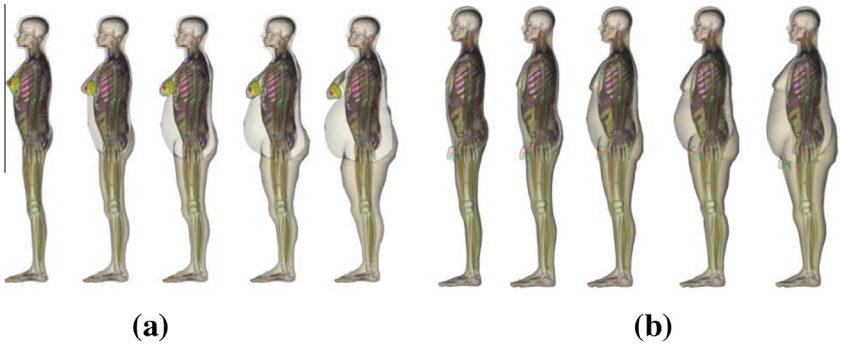

Fig. 2. Phantoms available in ARCHER. (a) RPI-Adult Females with body weights of 60, 74, 89, 100 and 122 kg. (b) RPI-Adult Males with body weights of 73, 86, 103, 117 and 142 kg.

adopted to fully utilize the tested multi-core CPU. Two variants of the code, $ARCHER_{GPU}$ and $ARCHER_{MIC}$, are developed specifically for the GPU and Intel MIC architectures. The serial code, including simulation scheduling, IO and user interactive parts is executed on the CPU. The photon transport kernel is delivered to GPUs and MICs for parallel execution using NVIDIA's CUDA C (NVIDIA, 2012) language and the Intel's OpenMP offload programming model, respectively. When the code is executed, the host code first reads in the geometry, cross-section and other user-defined parametric data and stores them in the host memory. It then selects an accelerator device, allocates memory on the device, and copies the data over. The host CPU then issues a command to the device to initiate the parallel transport simulation. Once the simulation is done, the dose results are transferred back to the host memory for further analysis and reporting.

Cross-section tables are stored in the texture memory to take advantage of the hardware-supported linear interpolation operations. Patient phantom data and tally arrays are kept in the global memory on device, while physical constants shared by all threads are kept in the constant memory.

One goal of the study was to carry out a fair performance comparison between multi-core CPU, GPU and coprocessor. To this end, we adopted the exactly same physics models, single floating point format and similar compiler options for different platforms. Besides, the same random number generator Xorshift (Marsaglia, 2003) was used in all the three code variants.

It is noticed that single precision (SP) floating point arithmetic is sufficient for CT dose calculations. For three variants of the codes, similar compiler options were applied for floating point arithmetic: "-ffast-math" for the CPU, "-use_fast_math" for the GPU, and "-fp-model fast = 2" for the coprocessor. For the CPU code, we used OpenMP to run multiple threads on a multiple-core CPU to make full use of its computing capability. Hyper-threading (HT) is enabled on the CPU which leads to an immediate 40% performance boost compared to the CPU code running the HT-disabled CPU.

2.2. Electron transport

Electron transport in ARCHER (Su et al., 2013) is based on the class-II condensed history method and continuously slowing down approximation. Møller scattering and Bremsstrahlung are modeled for interactions of certain energy-loss threshold.

Below the threshold, the energy losing is treated with CSDA. The effect of large number of elastic interactions in a given pathlength is modeled by Goudsmit and Saunderson (GS) multiple scattering theory (Goudsmit and Saunderson, 1940). Sempau's implementation (Sempau et al., 2000) is used to enable the electron to transverse multiple voxels in a single step, which speeds up the simulation significantly. The coupled electron-photon transport is handled explicitly; that is, all secondary particles produced in the primary particle transport are transported like primary particles.

2.3. Neutron transport

Neutron transport in ARCHER is currently limited to the calculation of the eigenvalue defined as the ratio of the number of fission neutrons in one generation to that in the previous generation. Our study considered two simple geometries: a bare spherical core and a binary slab system. One-speed model was used for elastic scattering, fission and capture. The absorption process was simulated using non-analog method. Weight-window technique composed of splitting and Russian roulette was employed to ensure the neutron always had an appropriate weight value. Collision and path-length estimators (kcol, kpath) were used to evaluate eigenvalues in each generation. The convergence of eigenvalue and fission source distribution was achieved by simulating a total of 1000 generations (the first 200 were inactive).

The GPU-based neutron code for eigenvalue calculations were developed using NVIDIA's CUDA C. The flowchart of the GPU-based code is shown in Fig. 3.

Recently we have developed an event-based vectorized MC code for the neutron eigenvalue calculations to investigate its effectiveness in solving the branch divergence problem on GPUs (Du et al., 2013).

The flow chart of vectorized MC method as implemented in ARCHER is shown in Fig. 4. We keep two particle stacks for storing the neutrons being simulated in the current batch. One stack, called F, is used to store neutrons that will undergo the flight analysis. The other one, called C, is used to store neutrons that will undergo the collision analysis. In the beginning, we put all the initial neutrons into the F stack, and perform the flight analysis for all the neutrons. After distance sampling, those neutrons that will travel without crossing medium interfaces will be stored in the C stack for later collision analysis, and those that travel across medium interfaces will move to the interface position and stay in F stack for another flight analysis. This process is repeated until the F stack is empty, when the collision analysis is executed for all neutrons in the C stack. At this point, a shuffling operation is applied to stack C to remove neutrons that are out of region of interest (ROI) and only keep survived ones for the collision analysis. Neutrons with weight values below some critical value after collision will be removed following a sampling process. The survived neutrons then enter the next loop of analysis.

By doing these iterative operations, we guarantee that all the neutrons being processed at the same time are undergoing the same physical events and involving the same sequence of instructions. This means that once the method is implemented on GPUs with each thread simulating one or more particles, all of the 32

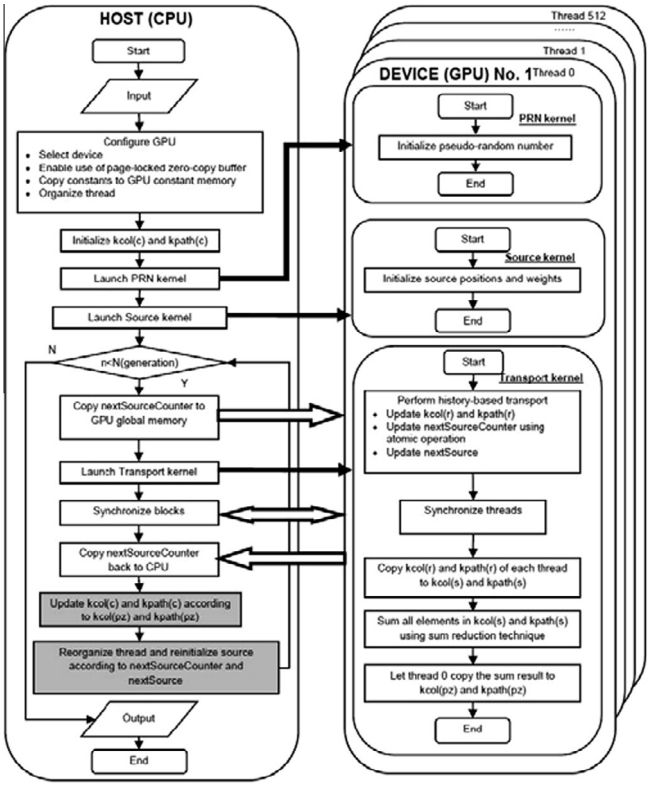

Fig. 3. Flowchart for the GPU-based eigenvalue calculation code.

```
Loop for Batches 1 … N
.
.       While  length(C) > 0
.       .
.       .       While  length(F) > 0
.       .       .       Free Flight
.       .       .       Shuffle F
.       .       .       Boundary Crossing Check
.       .       .       Shuffle F
.       .       .....
.       .
.       .       Shuffle C
.       .       Collision
.       .       Shuffle C
.       .....
.
....
```

Fig. 4. Simplified flowchart of GPU-based vectorized MC algorithm implemented in ARCHER.

threads within a warp will be executing the same instructions and the thread divergence does not occur. Different events are then executed iteratively in a sequential order until the storage stack becomes empty. Note that with the event-based method, there is no one-to-one correspondence between the particle history and GPU thread, since different portions of one particle history may be executed by different threads for the events within that history.

3. Applications and results

3.1. X-ray CT imaging dose

ARCHER was used to simulate X-ray CT imaging involving a modern multiple-detector CT scanner. ARCHER is coupled with a family of voxelized human phantoms for organ dose calculations. The main goal of this project is to test the feasibility of running an MC simulation for a CT protocol in "near real-time" with the aid of hardware accelerators.

We tested ARCHER-CT for a CT scan simulation using an abdomen phantom converted from the clinical CT images for a prostate cancer treatment (Ding et al., 2010) at Massachusetts General Hospital (MGH). To build the voxelized patient anatomical model, the contours of the tumor target (prostate) and other organs including rectum, bladder and femoral heads on CT images were designated by a radiation oncologist, and then combined to construct the voxelized phantom. The phantom contains $218 \times 126 \times 60$ voxels with a voxel size of $1.954 \times 1.954 \times 5\ mm^3$. Ten organs including prostate, rectum, bladder, bone and soft tissues are explicitly modeled. The CT images and a 3D vision of the constructed phantom are shown in Fig. 5.

The CT scan protocol is as follows: 20 mm collimation, 120 kVp beam and helical scan mode with a pitch of 1.375. A total of 10^8 photons were simulated for reaching satisfactory statistical precisions. Three variations of ARCHER-CT, including the CPU, GPU and MIC codes, were used respectively to calculate the doses for 9 organs. Combo physics mode was used for achieving both fast and accurate dose results. To validate the calculation, we compared the results with MCNPX (v2.5.0) using the exactly same simulation setup and tally card type 6 (F6: P). This can serve as an accuracy benchmark as the latter has been extensively validated in a wide range of applications including medical physics and nuclear engineering.

We obtained almost the same organ dose results from three code variants of ARCHER-CT, which is expected since the same algorithms were used by the codes. In Table 1 we show the organ dose results from the GPU version of ARCHER-CT and MCNPX as well as the percentage difference *diff* between the center values of corresponding results:

$$diff = \frac{\mathrm{DOSE}_{\mathrm{ARCHER\text{-}CT}} - \mathrm{DOSE}_{\mathrm{MCNPX}}}{\mathrm{DOSE}_{\mathrm{MCNPX}}} \times 100\% \quad (1)$$

It can be seen that the results from ARCHER-CT agree with results from MCNPX very well. For all the organs, the percentage differences from MCNPX are less than 2%. On the other hand, the differences are still considerably larger than the statistical errors of MCNPX results which are generally less than 0.5 percent. The discrepancy is presumably due to the approximations we made for various physics models and the absence of more detailed physics interactions like fluorescence. However, the accuracy achieved by ARCHER-CT is already sufficient for medical applications.

The simulation time of ARCHER-CT and MCNPX is listed in Table 2, from which it can be seen that all code variants of ARCHER-CT are significantly faster than MCNPX. In particular, the GPU code only takes 6.8 s to do the whole simulation while MCNPX takes 183 min. The GPU and MIC codes are seven times and twice, respectively, faster than the OpenMP based parallel code on CPU. There are several reasons why ARCHER-CT is many times faster than MCNPX: (1) MCNPX is a general purpose MC code which is designed to deal with complex physics models and complex geometry, thus the code complexity is also high. (2) Improved photon transport algorithms in ARCHER-CT, such as the use of cross section lookup tables, Woodcock delta tracking and fictitious cross section tables, make the simulation faster. (3) Emergent computing hardware facilities like the GPU and MIC provide significant computing power and further speed up the simulation.

To test the accuracy of ARCHER-CT, benchmark tests were performed against the production code, MCNPX which was run on Intel Xeon X5650 2.66 GHz CPU using a single thread. For a whole-body CT scan (142-kg RPI Adult Male and Obese Phantom, 120 kVp, 20 mm beam collimation and a pitch of 1), a total of 9×10^8 photons were simulated. Using single precision calculations and detailed physics, the average difference in organ doses

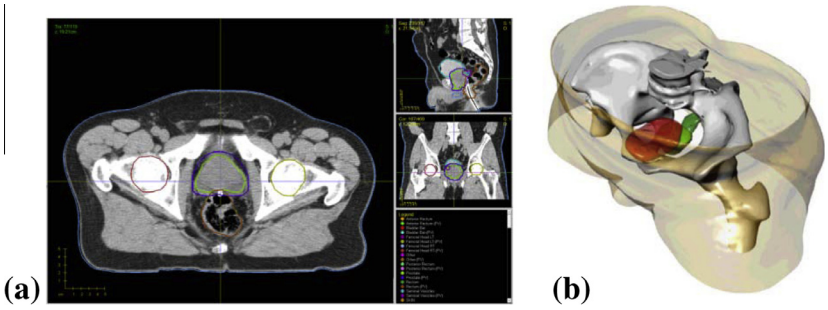

Fig. 5. (a) A patient-specific CT image for prostate cancer treatment. (b) The 3D vision of the constructed patient anatomy from segmented CT images.

Table 1
Organ dose results for the CT scan simulation with a clinical case based phantom. The first to third columns are MCNPX results, MCNPX relative standard deviation (RSD) and ARCHERCT results, respectively. The last column is the percentage difference between the results of ARCHER-CT and results for MCNPX.

Organ	MCNPX	MCNPX RSD	ARCHER	Diff
Anterior rectum	2.51E−07	0.40%	2.49E−07	−0.70%
Posterior rectum	2.70E−07	0.40%	2.72E−07	0.74%
Rectum	2.56E−07	0.39%	2.53E−07	−1.19%
Prostate	2.09E−07	0.32%	2.10E−07	0.46%
Bladder	2.71E−07	0.24%	2.72E−07	0.25%
Femoral head LT	5.02E−07	0.30%	5.00E−07	−0.48%
Femoral head RT	4.97E−07	0.30%	4.93E−07	−0.79%
Bone	4.68E−07	0.07%	4.63E−07	−1.13%
Soft tissue	3.11E−07	0.03%	3.06E−07	−1.56%

from $ARCHER_{GPU}$ and MCNPX was found to be 0.38%. Using simplified physics, the difference was found to be 1.9%, which is still excellent. With combo physics mode, the average difference is about 0.3%.

3.2. Electron transport

We tested the electron transport part of ARCHER in a heterogeneous slab geometry phantom. The phantom is a $30 \times 30 \times 30$ cm^3 cube with voxel size of $0.5 \times 0.5 \times 0.2$ cm^3. Along z axis, there are three levels: 2 cm water, 1 cm aluminum and 37 cm water. The source used in simulation is 20 MeV monoenergetic monodirectional parallel electron beam incident perpendicular to the x–y plane with a field size of 3.0×3.0 cm^2. A total of 6×10^6 histories were simulated. The depth dose of the central axis and lateral dose of different depth ($z = 1$ cm, $z = 4.8$ cm, $z = 7$ cm) were tallied and compared.

For comparison, we performed the same simulation using production MC code MCNPX and EGSnrc/DOSXYZnrc (Kawrakow and Rogers, 2000). The cutoff energies for electron and photon for all simulations are set to 200 keV and 50 keV, respectively. The results from MCNPX are very close to those from EGSnrc; for clarity only the latter are plotted here. Fig. 6 shows the central axis depth dose comparison. Fig. 7 illustrates the lateral dose for different depth.

It can be seen from Fig. 4 that at high dose region the ARCHER results are very close to EGS (within 1%). While the dose decreases with depth, the statistical error increases. At $z = 10$ cm the statistical error is about 10%, which is comparable with the difference between $ARCHER_{GPU}$ and EGS. All the voxels with doses greater than 50% of the dose maximum have statistical errors less than 1%. The difference between $ARCHER_{CPU}$ and $ARCHER_{GPU}$ is within 0.5% of the dose maximum (comparable with statistical error), which means that the adoption of single precision in $ARCHER_{GPU}$ does not introduce measurable error. Meanwhile, a very good agreement is observed between ARCHER and EGSnrc: For 99.1% voxels the difference of dose results between $ARCHER_{GPU}$ and EGSnrc is within 2% of the dose maximum.

Table 2
Simulation time (in seconds) of ARCHER-CT and MCNPX for the prostate phantom CT scan with 10^8 photon histories.

Code	$ARCHER_{CPU}$ 12 threads	$ARCHER_{GPU}$	$ARCHER_{MIC}$	MCNPX 12 threads
Time (s)	46.4	6.8	21.2	$1.1 \times E + 4$

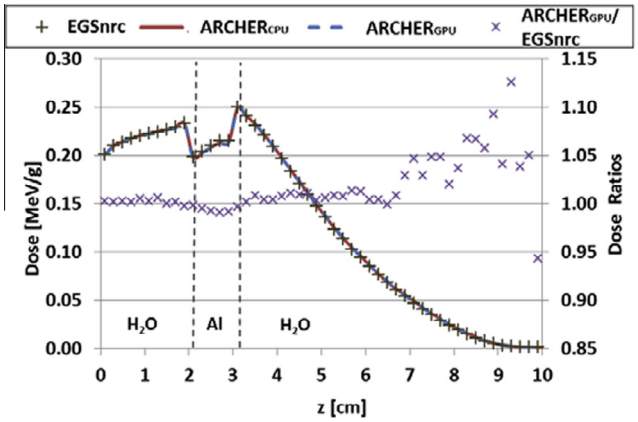

Fig. 6. Central axis depth dose curve and $ARCHER_{GPU}$/EGSnrc dose ratios of 20 MeV monoenergetic monodirectional parallel electron beam incident perpendicular to the x–y plane. Note that at the dose distal area (near $z = 10$ cm) the statistical error is in the order of 10%, which is comparable to the difference between $ARCHER_{GPU}$ and EGS.

The time used for simulating 6 million electrons is shown in Table 3. Note that the time listed here is the GPU kernel execution time, which excludes the I/O time on the host (2∼3 s and most of it is spent on reading phantom geometry). The results show that $ARCHER_{GPU}$ running on one NVIDIA M2090 GPU card is 4.3 times faster than $ARCHER_{CPU}$ running on one Intel X5650 CPU with 6 threads (six cores used). With six GPU cards, the simulation of 6×10^6 electrons can be done within 2 s, compared with 9213 s in MCNPX and 1645 s in EGSnrc, with single thread used. Besides the acceleration from the usage of GPUs, there are two other reasons why EGS and MCNPX are much slower than $ARCHER_{GPU}$. First, as general purpose codes, MCNPX and EGS have complicated code structures which handle a variety of application scenarios; this brings in some execution overhead. ARCHER is designed for medical use only and it is more concise and more efficient. Second, MCNPX and EGS employ more detailed physics models than ARCHER does, such as Doppler broadening and X-ray fluorescence. However, the relative simple physics models we used turn out to give sufficiently accurate results for most medical applications as shown above.

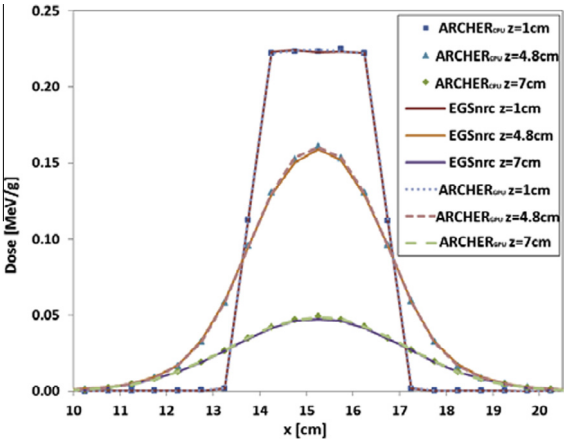

Fig. 7. Lateral dose curve at different depth of 20 MeV monoenergetic monodirectional parallel electron beam incident perpendicular to the *x*–*y* plane.

Table 3
Simulation time comparison for 6 million 20 MeV electrons incident on water-aluminum-water phantom.

Code used	Time [second]	Particles simulated per second
MCNPX	9213	6.51E+02
EGSnrc	1645	3.65E+03
ARCHER$_{CPU}$	31.6	1.90E+05
ARCHER$_{GPU}$ (1 card)	7.33	8.19E+05
ARCHER$_{GPU}$ (6 cards)	1.94	3.09E+06

3.3. Reactor eigenvalue calculations

ARCHER was applied to a neutron eigenvalue problem for nuclear reactor analyses. Here we present the results for a binary slab geometry setup where fuel and moderator are distributed alternatively. The parameters of the fuel slabs were: $\Sigma f = 0.01\ cm^{-1}$, $\Sigma a = 0.02\ cm^{-1}$, $\Sigma t = 0.1\ cm^{-1}$, $\nu = 2.5$, $\Delta x = 3.8$ cm, and the parameters of the moderator slabs were: $\Sigma a = 0\ cm^{-1}$, $\Sigma t = 0.1\ cm^{-1}$, $\Delta x = 10.0$ cm. The parameters were assigned such that the eigenvalues would finally be close to 1. A total of 10^6 initial neutron histories and 1000 generations were simulated by the CPU and GPU codes respectively. Double precision floating point arithmetic was used for guaranteeing the computation precision.

ARCHER$_{CPU}$ was run in serial mode using a single thread on an Intel Xeon X5650 CPU, while ARCHER$_{GPU}$ was run on a NVIDIA Tesla M2090 GPU card. For the whole simulation, ARCHER$_{CPU}$ takes 6077.5 s to finish and ARCHER$_{GPU}$ takes only 208.1 s. The GPU-based code is nearly 30 times faster than the CPU-based one. This shows the efficiency of using GPUs in accelerating the MC simulations for neutron transport.

3.4. Vectorized MC algorithms for GPUs

ARCHER was used to compare history-based and event-based MC algorithms for a neutron eigenvalue problem on GPUs. The geometry considered here is a heterogeneous 1-D system that consists of alternately distributed fuel and moderator slabs. A total of 10 fuel slabs and 11 moderator slabs are modeled. For simplicity we use the one speed approximation in our MC implementation. Three physical processes, elastic scattering, fission and capture, are being considered for each neutron history in the simulations, where the last two are regarded as absorption. The cross sections of each reaction are set such that the resulting eigenvalue is close to one. Specifically, we use $\Sigma F = 0.034\ cm^{-1}$, $\Sigma A = 0.08\ cm^{-1}$, $\Sigma T = 0.1\ cm^{-1}$, $\nu = 2.5$, $\Delta x = 3.8$ cm for the fuel, and $\Sigma A = 0.0001\ cm^{-1}$, $\Sigma T = 0.1\ cm^{-1}$, $\Delta x = 30.0$ cm for the moderator.

The kernel block size was set to be 256, and the number of neutrons simulated by each thread was 100. The grid size was then determined by dividing the total number of neutrons to be handled for the current kernel by 25,600. The values of these parameters were chosen so that the GPU code performance was optimum for our particular setup.

In Table 4, we show the running time of three different codes and the speed up factors relative to the CPU result. The ARCHER$_{CPU}$ code was tested on an Intel Xeon X5650 2.66 GHz CPU with 13G DDR3 memory. Only a single CPU thread was used and the code was run in sequential mode. The ARCHER$_{GPU}$ codes were tested on a NVIDIA Tesla M2090 GPU card.

The GPU-based vectorized Monte Carlo code, ARCHER$_{GPU}$ (vectorized), was found to be slower than its conventional GPU counterpart, ARCHER$_{GPU}$ (history-based), by a factor over 10. To find out the cause of the downgraded performance, the GPU execution statistics per neutron generation were collected and analyzed by using a profiling tool NVPROF. Fig. 8 displays the warp execution efficiency data of each kernel function, which is defined as the average number of active threads in a warp divided by the total number of threads in a warp (32). The magenta and green bars represent the GPU profiling data for conventional and vectorized algorithms, respectively. Numbers in the square brackets denote the number of times that each kernel is launched. The kernels that are used to initialize fission sites and neutron weights prior to the simulation of each generation in the conventional and vectorized codes are functionally the same; hence a similar efficiency. Except for that, all the kernels of the vectorized code have a higher efficiency than the single large transport kernel of the conventional code, because of the effectively decreased occurrence of divergent branches.

However, the advantage of higher warp execution efficiency is completely offset by the highly increased global memory transaction, as is illustrated in Fig. 9. Following the same convention, we use magenta and green bars to represent data for conventional and vectorized algorithms, respectively, and indicate the kernel launch counts in the square brackets after each kernel. Unlike in the conventional code where the neutron attribute data such as position, direction, energy, weight, etc. are created and consumed in the fast on-chip register space, in the vectorized code, they have to be frequently read from and written to the slow off-chip global memory, which is known to have a high access latency. For the problem considered in this study, the total global memory throughput per neutron generation of the vectorized code is on the order of 200 GB, which is ~60 times larger than that of the conventional code. The dramatically increased number of global memory transactions causes large amount of latencies on the GPU and makes the vectorized MC code much slower than the conventional one, although the former gives better warp execution efficiency.

Based on the test runs and profiling results, we can draw the preliminary conclusion that vectorized MC algorithm is probably not well suited for running on modern GPUs and the main reason is the high latency of global memory access. The requirement for frequent memory reading/writing is to a large extent intrinsic to the vectorized algorithm, so latency is most likely to continue to be a major issue for any effort of porting vectorized MC code to GPUs.

Table 4
Performance comparison between different transport codes in ARCHER.

Code	Computation time [sec]	Speedup
ARCHER$_{CPU}$	6077.5	1
ARCHER$_{GPU}$ (history-based)	208.1	29.2
ARCHER$_{GPU}$ (vectorized)	2278.9	2.7

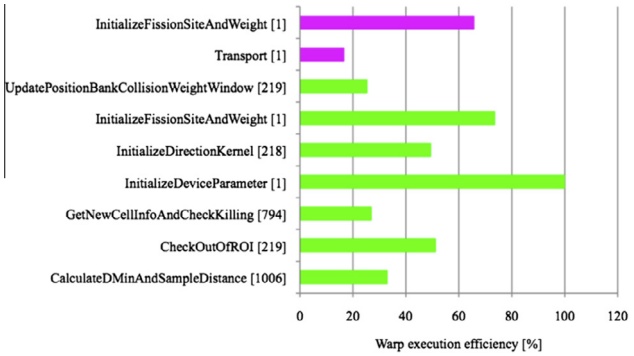

Fig. 8. Warp execution efficiency of kernel functions. Magenta and green bars represent the data for history-based and vectorized ARCHER$_{GPU}$ codes, respectively. Numbers in the square brackets denote the number of times that the kernel is launched. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

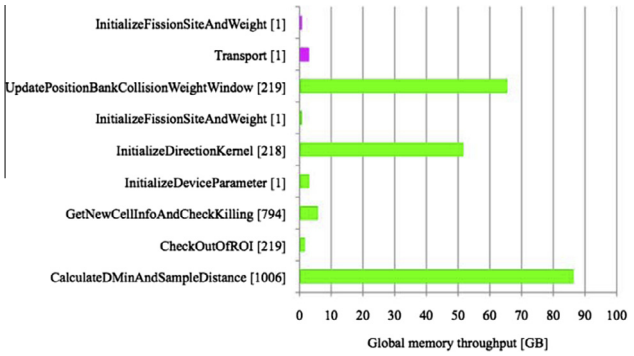

Fig. 9. Global memory throughput of kernel functions. Magnetic and green bars represent the data for history-based and vectorized ARCHER$_{GPU}$ codes, respectively. Number in the square bracket denotes the times that the kernel is launched. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Conclusion

Although still in its early stage of development, ARCHER has shown the utility as a versatile testbed for research in Monte Carlo software design under various emerging heterogeneous computing architectures. Preliminary tests for selected nuclear and radiological engineering problems focused on potential accelerations brought by ARCHER in comparison with general-purpose MC codes such as MCNP and EGSnrc. The performance comparison results showed that significant speed up can be achieved by using hardware accelerator such as GPUs and Intel coprocessors for MC radiation transport simulations. In general, the GPU-based code running on a NVIDIA Tesla M2090 and the MIC-based code running on an Intel Xeon Phi 5110p are 5–8 times and 2 times, respectively, faster than the OpenMP code running on an Intel Xeon X5650 CPU.

On the other hand, the actual performance of GPU and Intel Phi coprocessor is significantly lower than the theoretical peak performance. This is due to several reasons:

1. On GPUs the main issue is the problem called branch divergence: On the GPU, threads are split into groups called warps. Each warp contains 32 threads and executes one common instruction concurrently. Highest performance will be achieved if all the 32 threads of a warp agree on their execution path. If threads diverge at a certain data-dependent conditional branch, the warp will sequentially execute each branch. This phenomenon is called thread divergence. Because MC contains a large amount of conditional branches and which branch for a thread to enter is randomly selected, thread divergence becomes an inevitable problem affecting the overall performance.
2. Another factor affecting the efficiency is the memory latency. On GPUs, most of the data are put into the global memory due to their relatively big size. While being large in capacity, global memory has relatively high latencies compared with other memories such as registers and the shared memory. The data fetching and writing operations take long time to finish, and computation units have to wait for data to be ready, resulting in a low computing efficiency. On MICs, the similar case happens where the cores need to go through a loop topology to access the data in global memory, which takes long time for cores far from the memory.
3. The vector computing power of GPUs and MIC are not fully exploited. GPUs adopt the Single Instruction Multiple Threads (SIMT) architecture which is very similar to the Single Instruction Multiple Data (SIMD) architecture used by vector computers. Specially designed vectorized algorithms are needed to maximize the GPU performance. On MICs, each core is equipped with a 512-bit SIMD unit, making the whole card a powerful vector processor. One may rely on the compiler to utilize these SIMD units by using compiling options to automatically vectorize certain parts of the code. However, the effectiveness of such vectorization is often very limited. In order to take full advantages of the computing power of vector units, vectorized algorithms and dedicated SIMD programming are required. In ARCHER-CT, we used conventional history based algorithms for most applications and thus did not exploit the vector computing power of GPUs and MICs. For the neutron eigenvalue problem, we implemented an event-based vectorized MC algorithm for the eigenvalue problem, and found that its performance is even worse than the history-based algorithm. Based on the test runs and profiling results, we found that the main reason is the high latency of global memory access. The requirement for frequent memory reading/writing is to a large extent intrinsic to the vectorized algorithm, so latency is most likely to continue to be a major issue for any effort of porting vectorized MC code to GPUs.

There are possibilities to alleviate the global memory latency problem and improve the performance of the GPU code. First, in the flight analysis step, we keep executing the flight kernel until all of the neutrons enter the collision stack. As this process goes, the number of active neutrons simulated in the flight kernel is continuing to decrease and it could be well below one million for the last several flight kernel executions. In our GPU code, we always use a block size of 256 and let each thread simulate 100 neutrons, so if the total number of neutrons is only, say, tens of thousands, there are merely several hundred active threads for the GPU and they are much less than the maximal active concurrent threads on the Tesla M2090 card, which is 24576. The stream processors (SM) in GPU are not fully occupied, resulting in a very low efficient GPU usage. One possible solution is to adjust the block size and the number of neutrons per thread dynamically according to the current total number of neutrons being simulated to guarantee enough number of concurrent threads. We plan to implement this feature in the newer version of our code. Secondly, because the vectorized algorithms are effective in reducing thread divergence, it may be beneficial to address the global memory access problem directly. There are many "tried and true" programming techniques to hide the latencies for memory access, and some of those techniques may be useful for GPUs. For example, having sev-

eral stacks for both free-flight and collision operations may permit fetching one stack from memory while the GPUs are operating on another stack. Techniques such as prefetching, read-ahead/write-behind, asynchronous access, etc., have proven effective for conventional CPUs and may be effective in latency hiding for the GPU global memory access. These programming techniques, however, further complicate the Monte Carlo algorithms and may be highly dependent on relative speeds of specific hardware.

All these applications and interesting findings have allowed us to identify research issues that need to be further investigated. ARCHER is being refined by adding new features that are expected to lead to new Monte Carlo algorithms under various hardware/software platforms.

Acknowledgment

Research is supported in part by a grant from the U.S. National Institute of Biomedical Imaging and Bioengineering (R01EB0 15478).

References

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, 4878.

Baro, J. et al., 1995. PENELOPE: an algorithm for Monte Carlo simulation of the penetration and energy loss of electrons and positrons in matter. Nucl. Instrum. Methods Phys. Res. Sect. B 100 (1), 31–46.

Carrier, J.F. et al., 2004. Validation of GEANT4, an object-oriented Monte Carlo toolkit, for simulations in medical physics. Med. Phys. 31, 484.

Chen, W. et al., 2012. Fast on-site Monte Carlo tool for dose calculations in CT applications. Med. Phys. 39 (6), 2985.

Ding, A. et al., 2010. Monte Carlo calculation of imaging doses from diagnostic multidetector CT and kilovoltage cone-beam CT as part of prostate cancer treatment plans. Med. Phys. 37, 6199.

Ding, A. et al., 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.

Ding, T.L., Liang, C., Ji, W., Shephard, M.S., Xu, X.G., Brown, F.B., 2011. Evaluation of speedup of Monte Carlo calculations of simple reactor physics problems coded for the GPU/CUDA environment.: In ANS Mathematics & Computation Topical Meeting, Rio de Janeiro, RJ, Brazil.

Du, X., Liu, T., Ji, W., Xu, X.G., Brown, F.B., 2013. Evaluation of Vectorized Monte Carlo Algorithms on GPUs for a Neutron Eigenvalue Problem. In: International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013), Sun Valley, Idaho, USA.

Goudsmit, S., Saunderson, J., 1940. Multiple scattering of electrons. Phys. Rev. 57 (1), 24.

Gu, J. et al., 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.

Heimlich, A., Mol, A., Pereira, C., 2011. GPU-based Monte Carlo simulation in neutron transport and finite differences heat equation evaluation. Prog. Nucl. Energy 53 (2), 229–239.

Hissoiny, S. et al., 2011. GPUMCD: A new GPU-oriented Monte Carlo dose calculation platform. Med. Phys. 38, 754.

Jia, X. et al., 2011. GPU-based fast Monte Carlo simulation for radiotherapy dose calculation. Phys. Med. Biol. 56 (22), 7017–7031.

Jahnke, L. et al., 2012. GMC: a GPU implementation of a Monte Carlo dose calculation based on Geant4. Phys. Med. Biol. 57 (5), 1217.

Jia, X. et al., 2012. GPU-based fast Monte Carlo dose calculation for proton therapy. Phys. Med. Biol. 57 (23), 7783.

Kawrakow, I., Rogers, D., 2000. The EGSnrc code system. NRC Report PIRS-701, NRC, Ottawa, 2000.

Liu, T., Ding, A., Ji, W., Xu, X.G., 2012a. A Monte Carlo Neutron Transport Code for Eigenvalue Calculations on a Dual-Gpu System and Cuda Environment. In: Proceedings of International Topical Meeting on Advances in Reactor Physics (PHYSOR 2012), Knoxville, Tennessee, USA, April 15–20, 2012.

Liu, T., Ding, A., Ji, W., Xu, X.G., Carothers, C., and Brown, F.B., 2012b. A Monte Carlo Neutron Transport Code for Eigenvalue Calculations on a Dual-GPU System and CUDA Environment. In: International Topical Meeting on Advances in Reactor Physics (PHYSOR 2012). Knoxville, Tennessee, USA.

Liu, T., Ding, A., Xu, X.G., 2012c. GPU-based Monte Carlo methods for accelerating radiographic and ct imaging dose calculations: feasibility and scalability. Med. Phys. 39 (6), 3876.

Liu, T., Xu, G.X., Carothers, C.D., 2013. Comparison of Two Accelerators for Monte Carlo Radiation Transport Calculations, NVIDIA Tesla M2090 GPU and Intel Xeon Phi 3120 Coprocessor: a Case Study for X-ray CT Imaging Dose Calculation. Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo, Paris, France, October 27–31, 2013.

Marsaglia, G., 2003. Xorshift rngs. J. Stat. Software 8 (14), 1–6.

Nelson, A.G., Ivanov, K.N., 2010. Monte Carlo methods for neutron transport on graphics processing units using CUDA.: In PHYSOR 2010 –Advances in Reactor Physics to Power the Nuclear Renaissance. Pittsburgh, Pennsylvania, USA.

NVIDIA, 2012. CUDA C programming guide v5.0.

Pelowitz, D.B. (Ed.), 2011. MCNPX User's Manual, Version 2.7.0. Los Alamos National Laboratory.

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.

Su, L., Du, X., Xu, X.G., 2013. Monte Carlo Electron-Photon Transport Using GPUs as an Accelerator: Results for a Water-Aluminum-Water Phantom, in International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013), Sun Valley, Idaho, USA.

Su, L., Liu, T., Ding, A., Xu, X.G., 2012. A GPU/CUDA based Monte Carlo code for proton transport: preliminary results of proton depth dose in water. Med. Phys. 39 (6), 3945.

TOP500, 2013. http://www.top500.org.

Yepes, P.P., Mirkovic, D., Taddei, P.J., 2010. A GPU implementation of a track-repeating algorithm for proton radiotherapy dose calculations. Phys. Med. Biol. 55 (23), 7107.