RC25343 (AUS1301-004) January 14, 2013
Computer Science

IBM Research Report

Hardware Acceleration of an Efficient and Accurate Proton Therapy Monte Carlo

Thomas H. Osiecki, Min-yu Tsai*, Anne E. Gattiker, Damir A. Jamsek, Sani R. Nassif, W. Evan Speight, Cliff C. N. Sze
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758
USA

*National Taiwan University
Taipei, Taiwan

.

International Conference on Computational Science, ICCS 2013

Hardware acceleration of an efficient and accurate proton therapy Monte Carlo

Thomas H. Osiecki[a,*], Min-yu Tsai[b], Anne E. Gattiker[a], Damir A. Jamsek[a], Sani R. Nassif[a], W. Evan Speight[a], Cliff C. N. Sze[a]

[a]*IBM Research, Austin, TX 78758*
[b]*National Taiwan University, Taipei, Taiwan*

Abstract

Proton radiation therapy is one of the more effective forms of cancer treatment because of the high degree of selectivity afforded by the behavior of energetic protons in matter. But because radiation does not distinguish between tumor cells and healthy body tissue, it is important to insure that the radiation energy is deposited in the appropriate locations within a patient. This is even more important for proton beams because of the concentrated nature of the radiation energy dose they leave in a body. Predicting such dose distributions can be accurately done via complex and slow Monte Carlo based simulation (using tools such as Geant), but such simulators are too slow for use in interactive situations where a doctor is trying to determine the best beams to use for a particular patient. In this paper we report on an accurate but extremely fast Monte Carlo based proton dose distribution simulator code named Jack. The simulator uses the same physics as more complex tools, but leverages massive parallelization and a streamlined code architecture. The paper describes the state of Jack and shows runtime results for it with and without various hardware acceleration techniques. We benchmark Jack against Geant4.9.4.p01, a well established particle transport code, on a water phantom. Future plans are presented at the end for further speed enhancement and model development.

Keywords: proton therapy, cancer, massively parallel systems, GPU, POWER7, Monte Carlo

1. Introduction

Proton therapy for cancer treatment was first suggested by Robert R. Wison [1] in 1946. The first hospital-based proton therapy center was built in 1990 at the Loma Linda University Medical Center. Since that time proton therapy has rapidly expanded and been recognized as a premier solution for many types of cancer. At this time there exist eight centers in the United States alone.

The advantage of using protons for cancer therapy is in the way they interact with matter. Since protons lose energy at a rate proportional to the inverse of their kinetic energy, this leads to a strongly localized peak in energy deposition toward the end of its track, which is ideal for treating tumors deep in the body. Photon therapy on the other hand, deposits large amounts of energy toward the beginning of its track and this energy loss decreases as the photon approaches the tumor. This causes a very large dose of radiation to healthy tissue, which is undesirable.

[*]Corresponding author.
E-mail address: tosieck@us.ibm.com.

The treatment plan for an individual patient today is slow and largely done by hand. Given a medical image for a patient, an expert manually identifies the tumor location and determines the most promising radiation settings. These facilities have specific degrees of freedom in proton beam control. The settings, in general, are the proton beam energy and direction. They are chosen such that a medically useful dose is delivered to the tumor while leaving surrounding tissue minimally impacted. After the initial treatment plan is developed, it first must be approved by a medical doctor. Next, medical physicists test the calculated treatment plan by firing protons at a water phantom and take sophisticated measurements to compare to their expected dosage. A water phantom is a box of water that is bombarded by protons. This is a de facto standard in the medical physics community since the human body contains approximately 60% water. After measurements are complete, the patient can undergo treatment. The total delay from initial diagnosis to the beginning of treatment is on the order of weeks.

In addition to this delay the accuracy of the dose calculations is less than ideal due to widespread usage of analytical-based models [2][3][4], known as Pencil Beam Algorithms (PBAs), as opposed to a pure physics-based Monte Carlo. This is done simply because many of today's existing particle Monte Carlo transport codes are very slow and require many hours of runtime to achieve the required accuracy [5].

In order to completely revolutionize proton therapy and greatly improve treatment planning accuracy, the next generation of proton treatment planning must use fully accurate Monte Carlo based simulations. Those simulations in turn must do entire runs on the order of minutes, and the final optimization for beam energy and angle combinations must finish in minutes as well. These accomplishments are the key enablers of real-time proton therapy. Both aspects of this problem are in progress at IBM Research and this paper only focuses on the fast Monte Carlo.

At IBM Research we have realized the life-saving potential and research value of a fast and accurate Monte Carlo. In this paper we describe the state of Jack, our in-house proton therapy Monte Carlo. Our philosophy has been to "start from scratch" in order to avoid many problems that plague existing codes. By analyzing existing Monte Carlo codes and examining the literature, we have implemented and optimized the most accurate physical models and eliminated any overhead they may contain. Throughout the development process, separate code branches were created for other researchers to apply various hardware acceleration techniques. The results shown in this work are from a snapshot version of Jack. We plan additional future publications with the full version of Jack with hardware acceleration applied. To first order, adding additional models and changes will not have a large effect on the final results. Since we are validating Jack's accuracy, in this work we can turn Geant4 models on and off to do a true comparison.

Last, we compare and contrast existing particle transport Monte Carlos with respect to Jack.

2. Physics

Significant time has been spent understanding what physical models are most important to proton transport in matter. Therapy using protons gives us certain constraints that allow us to ignore physics not relevant to the problem at hand. For instance, proton therapy facilities generally use protons with a kinetic energy starting from around 70 to a maximum of 250 MeV/c. At these energies protons undergo electromagnetic and nuclear interactions. Proton electromagnetic interactions involve energy loss via ionization of electrons and multiple coulomb scattering for directional changes. The energy regime we are in allows us to ignore higher order energy corrections associated with energy loss calculations. The nuclear interactions consist of elastic and inelastic interactions. Inelastic models in general contain models for protons up to and above 1 GeV. This, for instance, involves many nuclear resonances that do not need to be modeled under our conditions. The models will be further described below.

2.1. Energy loss through ionization

Standard Bethe-Bloch theory is employed along with the most important corrections for density effects, shell corrections, the Barkas-Andersen correction and the higher order Bloch correction. This results in $\frac{dE}{dx}$ values calculated directly from well established theory. These values are tabulated, and interpolation is used for energy loss calculations. Liberated electrons travel $<<$ 1 mm, which is smaller than all voxel sizes in our geometry and so their kinetic energy is deposited immediately.

2.2. Fluctuations in energy loss

Fluctuations in energy loss can have large impacts on the resulting dosage distribution. For thin layers of material we employ the Urban model [6]. For thick absorbers the straggling function approaches the Gaussian distribution, which is also implemented in the Urban model.

2.3. Multiple Coloumb Scattering

Multiple coloumb scattering is implemented partially using Lewis theory [7] and an alternate correction for the tails of the distribution [8].

2.4. Elastic nuclear scattering

Protons can elastically scatter off the nuclei within a material. When this occurs, the final states (primary and recoil particle) are derived by sampling from tables of the cumulative distribution function of the centre-of-mass scattering angle, tabulated for a discrete set of kinetic energies. The CDF's are tabulated at 1 degree intervals and sampling is done using interpolation in energy and CDF values [9] [10]. The proton typically scatters off a heavy nucleus. The kinetic energy imparted to the nucleus is therefore treated like the electrons in ionization described above, in that the kinetic energy is deposited immediately due to their extremely short range. Occasionally the incident proton will liberate another proton, which is treated like a new particle and tracked.

2.5. Inelastic nuclear scattering

In inelastic scattering, the target nucleus is modeled using typical Fermi gas model statistics. The Fermi Energy and Momentum will determine what the distributions of the nucleons position and momentum are in the target nucleus. Once the nucleus is set up, we calculate the impact parameter for the incident proton straight throught the nucleus for each nucleon. If the impact parameter is less than a threshold value, related to the interaction cross-section, this signals a collision. After all the initial collisions between the incident and target nucleons are tabulated, we loop over these collisions looking for further collisions and any nuclear captures between nucleons that might occur in addition to propagating all nucleons due to their non-zero momentum. Once there are no more collisions to examine, all nucleons have either left the nucleus, been captured, or remain in the target and are thus untouched. If there is more than one captured nucleon, they are recombined into a heavier particle, typcially a deuteron, alpha particle or heavy ion depending on how many particles exist [11]. As with elastic scattering, any secondary heavy ions or particles will have their energy deposited locally. Protons that leave the nucleus are tracked. Current efforts focus on validating the inelastic scattering models of Jack and incorporating proper neutron kinematics.

3. Monte Carlo

3.1. Description

We chose to write the Monte Carlo in the C programming language for reasons of speed and staying power. Much of object oriented programming, classes in C++ for instance, produce many challenges when trying to host code on a hardware acceleration device. Also, we did not want to be constrained by any underlying infrastructure such as with Java and its JVM.

3.2. Water Phantom Results

Our initial test for validating Jack was on a simulated water phantom to compare to Geant4. The results shown in this section include all electromagnetic effects associated with protons, with nuclear effects to be integrated in the near future. In Figures 1, 2 and 3 we compare the total simulated integrated dose as a function of x, y, and z-position between Jack and Geant4. Examining the figures shows excellent agreement with Geant4 with no more than a maximum deviation of 2%, and the vast majority of bins agreeing to within 1%.

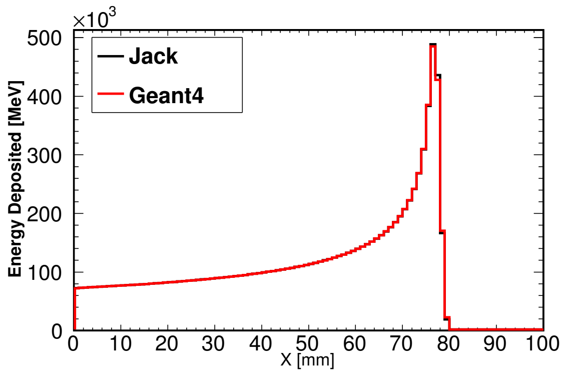

Fig. 1. Total integrated dosage (Electromagnetic interactions) calculation as a function of x-position for 100 MeV protons

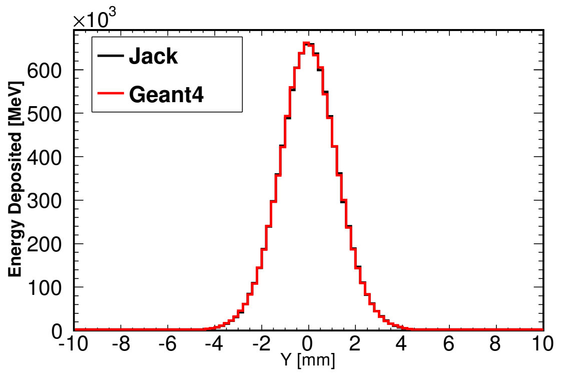

Fig. 2. Total integrated dosage (Electromagnetic interactions) calculation as a function of y-position for 100 MeV protons

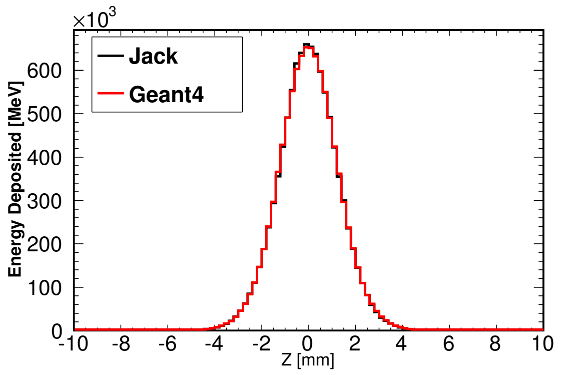

Fig. 3. Total integrated dosage (Electromagnetic interactions) calculation as a function of z-position for 100 MeV protons

4. Hardware Acceleration

In the field of high-performance computing (HPC), the use of hardware accelerators such as GPUs, FPGAs and multi-core processors is standard practice. GPUs-as-accelerators for compute-intensive work other than graphic processing (GPGPUs or General-purpose GPUs) consist of hundreds to thousands of small cores designed for parallel computation. Computationally-intensive parallel workloads can achieve high performance utilizing a commodity PC by taking advantage of the GPU's high floating-point throughput and memory bandwidth.

The Open Computing Language (OpenCL) is a framework that enables programmers to access various accel-

Table 1. Tabulated runtime results of Jack for various hardware platforms using a total of 10,000 simulated protons

Platform	Step Size (mm)	Voxel Size (mm^3)	Energy (MeV)	Execution time(s)	Speedup
i7	0.01	1x1x1	100	25.42	1x
	0.05	1x1x1	100	5.09	1x
	0.1	1x1x1	100	2.56	1x
	0.01	2x2x2	100	25.40	1x
	0.05	2x2x2	100	5.09	1x
	0.1	2x2x2	100	2.56	1x
	0.01	1x1x1	200	85.94	1x
	0.01	2x2x2	200	86.93	1x
i7 multi-core (8)	0.01	1x1x1	100	5.08	5.0x
	0.01	1x1x1	200	17.53	4.9x
	0.01	2x2x2	100	5.64	4.5x
	0.01	2x2x2	200	16.71	5.2x
Power P730	0.01	1x1x1	100	0.75	33.9x
Core 2 OpenCL	0.01	1x1x1	100	7.70	3.3x
	0.01	2x2x2	100	7.69	3.3x
GPU (GTX 295)	0.01	1x1x1	100	0.68	37.57x
	0.01	1x1x1	200	2.00	42.91x
	0.01	2x2x2	100	0.67	37.91x
	0.01	2x2x2	200	2.03	42.80x

erators in a C-based unified way without proprietary graphics programming techniques sometimes used in GPU programming or writing lower-level HDL (hardware description language) constructs for FPGAs.

To understand the performance potential of Jack on different hardware platforms, we have conducted a number of experiments. In this section, we report our results and compare the performance between several implementations.

4.1. *Platforms and Hardware*

We initially experimented with Jack in two multi-core systems, both as a single-threaded and parallel application utilizing the pthreads library. The first system is an Intel Core i7-920 CPU running at 2.66 GHz with an 8 MB L2 cache and 6 GB of main memory. There are 4 cores, each with 2 hardware contexts for a total of 8 hardware threads. This same system was then used to run Jack using the pthread library with 2, 4 and 8 threads. For greater pthread scalability tests, we ran the parallel Jack on an IBM Power P730 box with 2 POWER7 chips running at 3.875 GHz with 48 GB of main memory. Each chip has 8 cores, and each core has 4 independent hardware threads for a total of 64 independent hardware contexts.

To explore the speedups possible via GPU execution, we experimented with the OpenCL Jack version of the code on a system comprised of an Intel Core 2 Quad CPU running at 2.66 GHz with a 4 MB cache and 4 GB of DRAM. This system also contained an NVIDIA GeForce GTX 295 GPU card with 240 GPU cores running at 1.24 GHz and a total of 896 MB of memory. We are aware that there exist other GPU devices that promise much greater performance, and we intend to explore the latest technology as we are able to procure newer devices.

4.2. *Implementation*

For Jack with OpenCL, a CPU works as the host and is responsible for the initialization process, which then transfers the data into OpenCL kernels. These OpenCL kernels are then run on the CPU or GPU and execute the Monte Carlo proton algorithm described above. To reduce average memory access latency, all possible structures (e.g., the current state of a proton) and variables are moved into local or private memory due to the long latencies associated with access global memory in the GPU.

Since there is no built-in random number generator function in OpenCL, we provide one in the Jack code. This functionality also runs in an OpenCL kernel to generate random numbers and normal distribution values.

Table 2. Tabulated runtime results of Jack for various hardware platforms using a total of 1,000,000 simulated protons

Platform	Step Size (mm)	Voxel Size (mm^3)	Energy (MeV)	Execution time (s)	Speedup
i7	0.01	1x1x1	100	2542.49	1x
	0.05	1x1x1	100	509.10	1x
	0.1	1x1x1	100	256.05	1x
	0.01	2x2x2	100	2540.49	1x
	0.05	2x2x2	100	509.16	1x
	0.1	2x2x2	100	255.33	1x
	0.01	1x1x1	200	8595.65	1x
	0.01	2x2x2	200	8694.67	1x
i7 multi-core (8)	0.01	1x1x1	100	488.94	5.2x
	0.01	1x1x1	200	1653.01	5.2x
	0.01	2x2x2	100	488.55	5.2x
	0.01	2x2x2	200	1672.05	5.2x
Power P730	0.01	1x1x1	100	65.0	39.1x
Core 2 OpenCL	0.01	1x1x1	100	635.62	4.0x
	0.01	2x2x2	100	651.41	3.9x
GPU (GTX 295)	0.01	1x1x1	100	39.69	64.05x
	0.01	1x1x1	200	132.26	64.99x
	0.01	2x2x2	100	39.64	64.09x
	0.01	2x2x2	200	132.46	65.64x

The initial random seeds are generated by the host cpu and then transferred into the appropriate kernel for each particle. Math functions with the *native_* prefix are invoked in the kernel if available for performance reasons.

Data transfer between the host and device memory can frequently be a bottleneck in GPU computation. As a result, we minimize data transfer as much as possible. The record of the particle's individual steps (constituting a "track") tests the limits of memory capacity, necessitating particles to be divided into "batches" to prevent running out of resources. For example, over 500 MB is needed for the tracking records for a quarter of a million particles. We typically divide the job into four batches for one million protons to avoid memory access related problems.

4.3. Results

Tables 1 and 2 summarize the experimental results for 10,000 and 1,000,000 protons. We vary certain parameters in Jack that have a high impact on overall performance. The step size, for example, has an almost linear effect on run time. By default we choose a step size of 0.01 mm to achieve suitably accurate results. Using higher step sizes will change the resulting distribution and will result in a degradation in the accuracy of the results. The voxel size of our geometry was found not to be a large factor in performance, even though we split our data collectors among threads and then perform a reduction at the end of a run. Also, the kinetic energy has a non-linear scaling effect on the performance of the Jack code. To put the results in Tables 1 and 2 into perspective, a single-threaded run of Geant4 (only Electromagnetic Interactions for a proper comparison) for 1,000,000 protons with a step size of 0.01 mm takes on the order of 38 *hours* to complete on an i7, resulting in a maximum 3446x speedup for Jack across all architectures examined here.

4.3.1. Multi-threaded (pthread) Results

Fig. 4 presents speedup results when utilizing pthreads on the i7 for up to 8 threads. A maximum speedup of 5.2 is achieved when using all 8 hardware thread contexts. Note that speedup is linear through 4 threads, after which time threads are forced to share some CPU resources (due to the hyperthreaded architecture of the Intel i7 core) and resulting in worsening parallel efficiency. Similarly, the P730 results scale linearly up to 32 threads, then efficiency begins to fall off as the 4 threads compete for some CPU resources such as a shared pipeline, floating point units, etc. The 16-core Power P730 achieves a nearly 40x performance improvement over the initial Jack implementation for a 1M particle-sized problem.

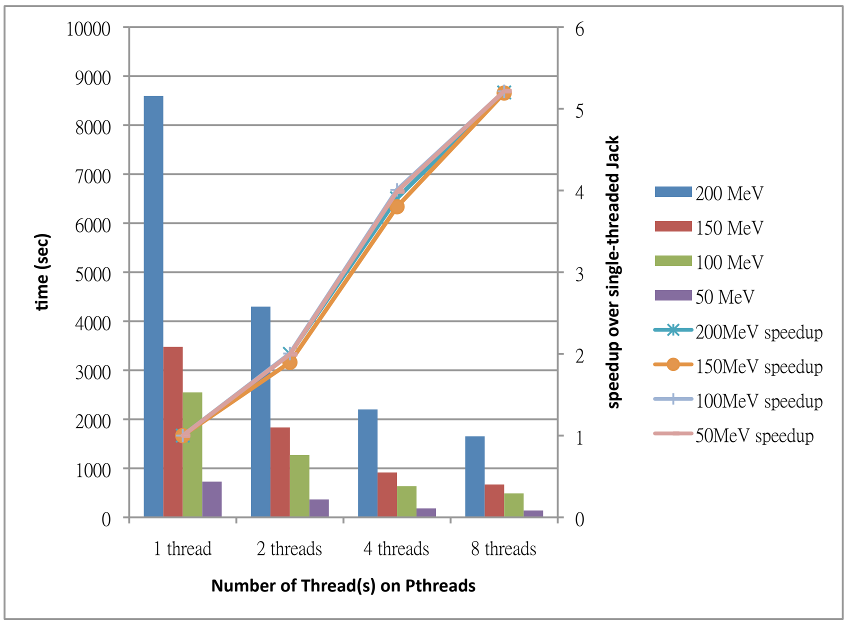

Fig. 4. Performance of Multi-threaded (pthread) Jack on i7 CPU.

4.3.2. Scaling with Problem Size

Fig. 5 depicts the performance of the multi-threaded version of Jack on 8 threads for two different step sizes and three different problem sizes. As the number of particles increases, the scalability of the application improves because there is more work available for each thread, reducing fixed-cost overhead of the final synchronization and reduction operations. At smaller problem sizes, smaller step sizes also serve to provide more work for each thread, improving scalability as shown by the .01 mm vs. .1 mm bars.

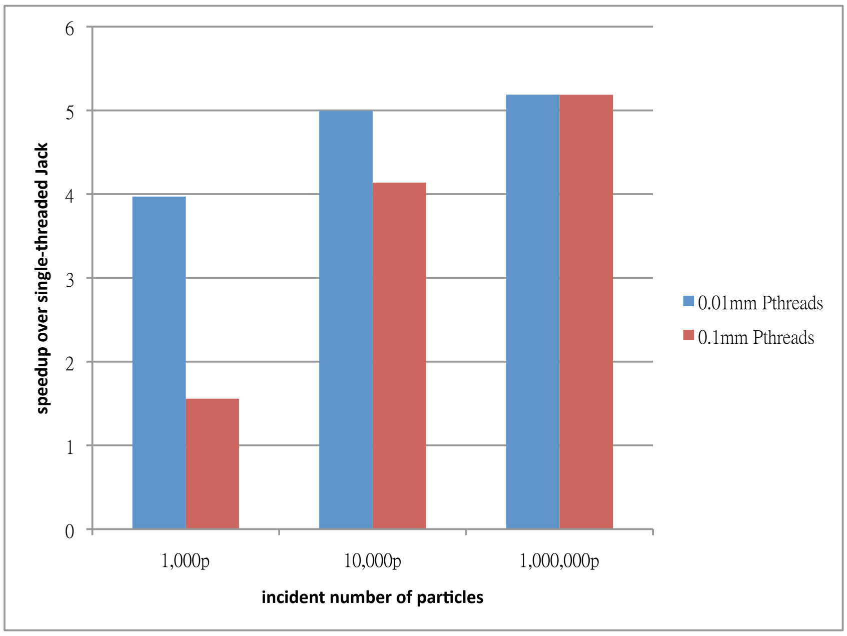

Fig. 5. Speedup of i7 Multi-threaded Jack (8 threads) as Step Sizes and Problem Sizes Change.

4.3.3. *Core 2 OpenCL vs. GPU OpenCL*

Although the maximum frequency of the GPU is less than half that of the host Core 2 CPU, Fig. 6 shows that the massive parallelism available by using the GPU enables a 10-fold performance improvement when running the OpenCL code on both processing elements with a large number of protons. Because the simulation is embarrasingly parallel, Jack is very amenable to the type of parallelism offered by GPU systems. There are 30 compute units in the NVIDIA GTX 295 per core, and 8 streaming processors per compute unit for a total of 240 streaming processors in a single core of an NVIDIA GTX295. For the Intel Core 2 CPU, there are a total of 4 compute units running at twice the frequency of the GPU units. The GPU is able to support thousands of active threads per multiprocessor, and Fig. 6 shows that it is well suited for running large-scale and highly parallelizable code such as Jack.

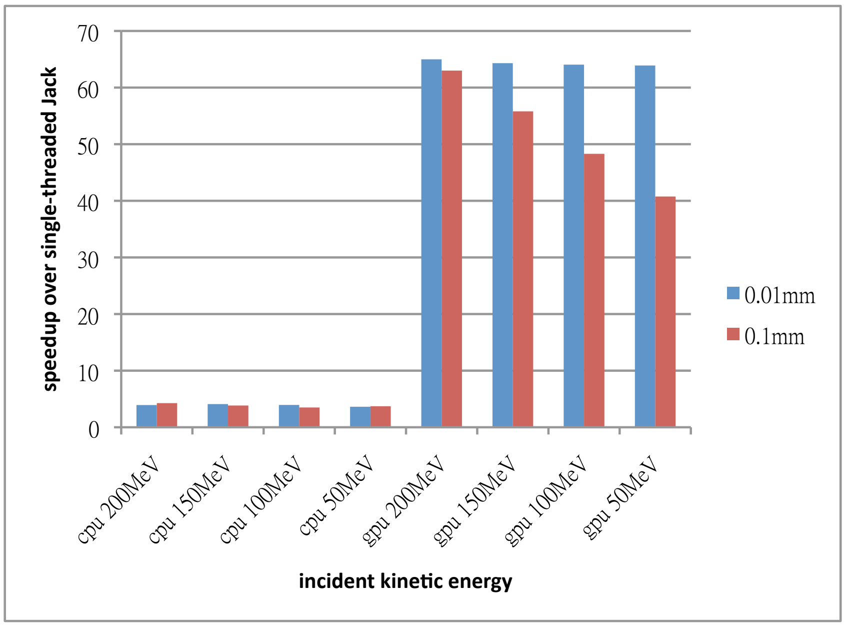

Fig. 6. Performance of a 1 Million Particle Execution on Host Core 2 CPU vs. NVIDIA GPU.

4.3.4. *GPU*

With a million incident particles, the NVIDIA GPU achieves a speedup of around 64x that of the single-threaded Jack. Unsurprisingly, the larger the number of particles, the more parallelizable work can be distributed to the streaming processors of the GPU adding both improved performance and improved accuracy. Fig. 7 demonstrates this effect over various step and problem sizes on the GPU. Tolerating memory latency is achieved via utilizing a high degree of multithreading, e.g., *warps* in the GPU. A warp is a group of threads executed together to take advantage of data processed in a SIMD (single-instruction multiple-data) fashion by the GPU. When one warp stalls due to a memory access, processors choose another ready warp to work on. Switching the workload from stalled warps to ready warps repeatedly keeps the streaming processors busy and parallel efficiency high.

5. Related Work

5.1. *Geant4*

Geant4 is a toolkit for the simulation of the passage of particles through matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science [12] [13]. It is the successor of the GEANT series of software toolkits developed by CERN, and the first to use object

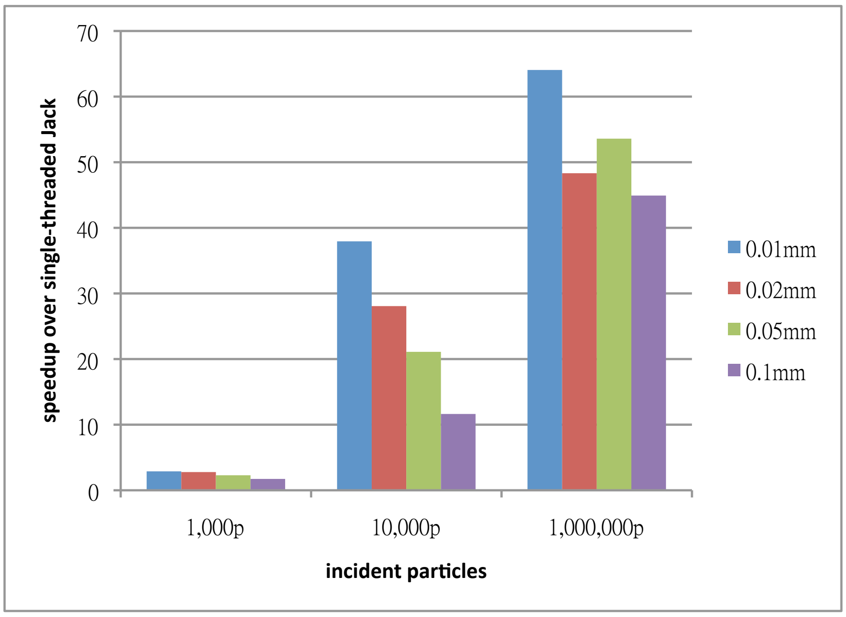

Fig. 7. Performance Relative to Single-threaded Jack for Various Dosage Samples and Step Sizes on the GPU.

oriented programming (in C++). Its development, maintenance and user support are undertaken by the Geant4 Collaboration. The software is used by a number of research projects around the world.

Geant4 includes facilities for handling geometry, tracking, detector response, run management, visualization and user interface. For many physics simulations, this means less time needs to be spent on the low level details, and researchers can start immediately on the more important aspects of the simulation.

The scope of Geant4's capabilities makes it a very powerful tool but as this work has shown, it has significant speed drawbacks when it comes to proton therapy. These drawbacks could be mitigated if Geant4 could be easily hosted on a GPU, which is not the case.

5.2. MCNPX

MCNPX is a general-purpose Monte Carlo radiation transport code for modeling the interaction of radiation with matter. MCNPX stands for Monte Carlo N-Particle eXtended. It extends the capabilities of MCNP4C3 to nearly all particles, nearly all energies, and to nearly all applications without an additional computational time penalty. MCNPX is fully three-dimensional and time dependent. It utilizes the latest nuclear cross section libraries and uses physics models for particle types and energies where tabular data are not available. Applications range from outer space (the discovery of water on Mars) to deep underground (where radiation is used to search for oil.) MCNPX is used for nuclear medicine, nuclear safeguards, accelerator applications, homeland security, nuclear criticality, and much more. MCNPX is written in Fortran 90, runs on PC Windows, Linux, and unix platforms, and is fully parallel (PVM and MPI) [14]. MCNPX still suffers much overhead from a large framework designed to apply to most situations as oppose to our focus on proton therapy. We expect similar speedups from Jack with respect to MCNPX.

5.3. FDC

The Fast Dose Calculator (FDC) [15] is a Monte Carlo track-repeating algorithm based on Geant4. FDC was developed to increase computation speed without diminishing dosimetric accuracy. The algorithm used a database of proton trajectories in water to calculate the dose of protons in heterogeneous media. The extrapolation from water to 41 materials was achieved by scaling the proton range and the scattering angles. The scaling parameters were obtained by comparing Geant4 dose distributions with those calculated with FDC for homogeneous phantoms. The FDC algorithm was tested by comparing dose distributions in a voxelized prostate cancer patient as calculated with well-known Monte Carlo codes (Geant4 and MCNPX). The track-repeating approach reduced the CPU time required for a complete dose calculation in a voxelized patient anatomy by more than two orders of magnitude, while on average reproducing the results from the Monte Carlo predictions within 2% in terms of dose and within 1 mm in terms of distance. Since it is not a true Monte Carlo based simulation it cannot guarantee the same usability and control over modeling.

5.4. SMC

The National Cancer Center hospital in Japan have implemented a simplified Monte Carlo (SMC) [16]. Though it has relatively good accuracy, speed and has a home-grown philospthy that is close to ours, the SMC could modify some models for more accuracy. It uses the Highland formula for multiple Coloumb scattering which has been shown to have issues in the tails [17] in addition to ignoring absorption and lateral scattering due to nuclear reactions. We intend Jack to be usable for both clinical and research applications in medical physics and so the proper incorporation and ability to separate different models from each other is important.

6. Conclusion

IBM Research has implemented a fast and accurate in-house Monte Carlo for proton therapy called Jack. It has been written from the ground up in order to capture all the physics necessary for clinical work and research applications in proton therapy. We verified that Jack is accurate by comparing to Geant4 using the electromagnetic interactions of protons. Also by hosting Jack on a GPU, we achieve on the order of 3446x speedup over Geant4 and using a Power P730 box we achieve a 2104x speedup.

Future work involves completing the validation of inelastic nuclear models in Jack, in addition to more effort on the hardware acceleration side, which in future publications will show results with both electromagnetic and nuclear interactions.

Acknowledgements

The authors would like to thank our colleagues Xiaodong Zhang and Dragan Mirkovic at MD Anderson Cancer Center, Pablo Yepes at Rice University, and Wayne Newhauser at Louisiana State University for invaluable discussions related to this work.

References

[1] Robert R. Wilson, Radiological Use of Fast Protons, Radiology 47 (1946).
[2] Petti P. L., Differential-pencil beam dose calculations for charged particles, *Med. Phys. 19* (1992) 137-49.
[3] Hong L. *et al.*, A pencil beam algorithm for proton dose calculations, *Phys. Med. Biol. 41* (1996) 1305-30.
[4] Szymanowski H. *et al.*, Experimental determination and verification of the parameters used in a proton pencil beam algorithm, *Med. Phys. 28* (2001) 975-87.
[5] Pagnetti H. *et al.*, Clinical implementation of full Monte Carlo dose calculation in proton beam therapy, *Phys. Med. Biol. 53* (2008) 4825-53.
[6] Lassila-Perini K., Urban L., Energy loss in thin layers in GEANT, Nucl. Instr. Meth. Phys. Res. *A362* (1995) 416.
[7] Lewis H. W., Phys. Rev. *78* (1950) 526.
[8] Urban L., A multiple scattering model in Geant4, Preprint CERN-OPEN-2006-077 (2006).
[9] Fesefeld H., Simulation of Hadronic Showers, Physics and Applications.” Technical Report PITHA 85-02, Aachen, Germany 1985.
[10] Degtyarenko P. V., Kossov V. M. and Wellisch J. P. Eur. Phys. J. A 8 (2000) 217
[11] Folger G., Ivanchenko V. N. and Wellisch J. P. Eur. Phys. J. A *21* (2004) 404-17.
[12] Agostinelli S. *et al.*, GEANT4: a simulation toolkit, *Nucl. Instrum. Methods. Phys. Res. A 506* 250-303.
[13] Allison J. Geatnt4 developments and applications, IEEE Transactions on Nuclear Science *53* No. 1 (2006) 270-278.
[14] Waters L. MCNPX user’s manual version 2.4.0, *Los Alamos National Laboratory Report* LA-CP-02-408 (2002).
[15] Yepes P., Randeniya S., Taddei P., and Newhauser W. Monte Carlo fast dose calculator for proton radiotherapy: application to a voxelized geometry representing a patient with prostate cancer. *Phys. Med. Biol. 54* (2009) N21-8.
[16] Hotta K. Improved dose-calculation accuracy in proton treatment planning using a simplified Monte Carlo method verified with three-dimensional measurements in an anthropomorphic phantom, *Phys. Med. Biol. 55* (2010) 3545-3556.
[17] Beringer J. *et al.* (Particle Data Group). Review of Particle Physics, *Phys. Rev. D*86, 010001 (2012).