Monte Carlo dose calculation using a cell processor based PlayStation 3 system

James C L Chow¹,², Phil Lam¹ and David A Jaffray¹,³

¹ Department of Radiation Oncology, University of Toronto and Radiation Medicine Program, Princess Margaret Hospital, University Health Network, Toronto, Ontario M5G 2M9, Canada; ² Department of Physics, Ryerson University, Toronto, Ontario, Canada; ³ Department of Medical Biophysics, University of Toronto, Ontario, Canada. james.chow@rmpuhn.on.ca

Abstract. This study investigates the performance of the EGSnrc computer code coupled with a Cell-based hardware in Monte Carlo simulation of radiation dose in radiotherapy. Performance evaluations of two processor-intensive functions namely, HOWNEAR and RANMAR_GET in the EGSnrc code were carried out basing on the 20-80 rule (Pareto principle). The execution speeds of the two functions were measured by the profiler gprof specifying the number of executions and total time spent on the functions. A testing architecture designed for Cell processor was implemented in the evaluation using a PlayStation3 (PS3) system. The evaluation results show that the algorithms examined are readily parallelizable on the Cell platform, provided that an architectural change of the EGSnrc was made. However, as the EGSnrc performance was limited by the PowerPC Processing Element in the PS3, PC coupled with graphics processing units or GPCPU may provide a more viable avenue for acceleration.

1. Introduction

To date, scientific computing is conducted by general-purpose computer processor (CPU) because of its relatively low cost. However, with the recent rapid development of non-traditional computer processor architectures driven mainly by the needs of the computer graphics and gaming industries, these architectures become popular. An example is the Cell processor used in the video game graphics and scientific computing.¹² The Cell Processor was designed to be a hybrid of conventional processors (e.g. Opteron, Core 2) and specialized processors such as graphics processing units (GPUs). Jointly developed by Sony Computer Entertainment, Toshiba and IBM (known collectively as STI) development started in 2001 and spanned 4 years, at a total cost of approximately $400 million. The first major commercial application of the Cell was the PlayStation3 (PS3) game console. The low cost of the console has made it an interesting choice for scientific computing on a budget. A notable example is Gaurav Khanna's “PS3 Gravity Grid,” (http://gravity.phy.umassd.edu/ps3.html) which is used to solve problems in astrophysics.

The EGSnrc Monte Carlo code⁵⁶ is used commonly to provide accurate dosimetric calculation of kV-MV photons and electrons for inhomogeneous medium in radiotherapy.⁷¹⁰ Since energy deposition or dose from each photon or electron particle is by definition statistically independent as well as governed by the same process,¹¹ such Monte Carlo simulations can be classified as "embarrassingly parallel" problems when implemented. The parallelizability of EGSnrc, coupled with the low cost of Cell-based hardware and the success of other efforts in similar fields suggested that an investigation into the possibility of
porting EGSnrc to a Cell-based platform would be worthwhile. Therefore, this study is aimed at (1) to determine whether EGSnrc is a suitable candidate for parallelization on the Cell platform; and (2) to recommend software architecture most suited to accomplish and implement.

2. Methods
2.1 Tools
2.1.1 PS3
In the PS3, the Cell processor has 9 cores, 8 of which are homogeneous as shown in Figs. 1(a) and 1(b). The main processor is known as the PowerPC Processing Element, or PPE. The remaining 8 processors are known as Synergistic Processing Elements (SPEs). Each core is completely independent, containing a processing unit, local cache and DMA module to access the main memory, which is shared. This is possible through the Element Interface Bus (EIB), which provides 204.8GB/s of bandwidth. The Cell processor in the PS3 contains one SPE that is disabled for yield purposes, and reserves another for OS use, leaving a total of 6 SPEs for program use.

The instruction set is a superset of the IBM PowerPC set. In fact, the PPE is binary-compatible with the PowerPC 970 architecture. This is a boon to developers, as it will allow any application compiled for this architecture to run natively on the Cell. It also allows for applications to be developed in steps. For instance, an algorithm can be written in the conventional fashion, then tested and benchmarked on the Cell processor without having to deal with the complexity of parallelization. Once the algorithm has been verified, it can be converted into a parallel algorithm taking advantage of the SPEs and other features of the Cell not present in the PPC970 architecture. This separates algorithm design, verification and acceleration into separate stages, simplifying the development process.

2.1.2 Linux system and compilers
In this study, Linux operation system called “Yellow Dog Linux”, or YDL for short developed by Terra Soft Solutions, was used as the programming platform. It is based on CentOS, and is designed to support the PS3/Cell. YDL features kernel support for all of the hardware available on the Cell, including PPE and SPEs.

There are two sets of compilers available for the Cell processor running Linux. IBM distributes its XL series of compilers for Fortran and C. There are also Open Source GNU compilers available for Fortran and C/C++. Both of these produce separate executables for the PPE and SPEs, combining them into a single program image for execution. Traditional optimization technique is used, though the IBM XL compilers generally produce speedier executables.

2.2 Performance evaluation
Performance evaluation was carried out on small portions of the EGSnrc code. These portions were chosen following the informal 80-20 rule (Pareto principle) of large systems. The rule states that a
A typical program spends 80% of its time executing 20% of the code. The reasoning was that the architecture code surrounding the essential algorithms could be adapted regardless, and it was only the core of the program that needed to be evaluated. Two processor-intensive functions, as shown in Fig. 2, named HOWNEAR and RANMAR_GET were selected in the performance test. These functions collectively account for over 20% of execution time, while comprising less than 1% of the total number of lines of code. However, it should be noted that such evaluation only tests if the whole EGSnrc code is worthwhile to parallelize. It is because a speed-up of 1.2 based on the Amdahl’s law shows that only parallelizing these two functions is inadequate. \[ p \leq \frac{1}{1 + f \cdot (p - 1)} \]  
where \( f \) is the fraction of the work of the program that must be done in serial mode. The execution speed of these functions was measured in situ during execution of the original program using the profiler gprof, which specified the number of executions and the total time spent in the function, allowing for an estimate. The functions were then re-implemented in a testing architecture designed using the ideas discussed above. This proof-of-concept provided identical functionality as the original, serial functions, but did so in a parallel, accelerated way.

**Figure 2.** Pie chart showing the proportion of time each of the major functions takes as part of the whole with RANMAR_GET and HOWFAR highlighted.

HOWNEAR is employed to calculate the minimum distance of the current particle to a voxel boundary through a series of logical comparisons. RANMAR, on the other hand, is the random number generator used in EGSnrc. It is responsible for generating a reproducible, independent random number stream. These key algorithms were isolated from the EGSnrc code and re-implemented in a test framework. This framework involved dividing the program functionality in two - isolating branching code for execution on the PPE and allowing the SPEs to perform the requisite calculations. Execution began as a single thread on the PPE, which then spawned a number of child threads as shown in Figs. 3(a) and 3(b). Each thread made a series of decisions before queuing for access to an SPE. This was implemented in an asynchronous, time-slice based multithreaded model, with each PPE thread competing for an available SPE. Locks were used to synchronize SPE resources; a given PPE thread could queue its' task with a 'captured' SPE once it acquired its' corresponding lock by passing mailbox messages containing a pointer to instructions and data. The SPE, upon receiving this message, retrieved the data in question, executed the function and wrote the result back, informing the calling thread that the operation was completed. This process was self-managed (i.e. each thread would compete for available resources) via conventional locking mechanisms. For testing, the number of available SPEs and the size of the task were varied. Additionally, the "size" of the task performed by the SPEs was varied by changing the number of
times the function was executed per request. This was represented using the variable TASK_SIZE; for TASK_SIZE=1, the function (RANMAR_GET or HOWNEAR) was executed only once per request made by a PPE thread. For TASK_SIZE=10, the function was looped ten times, the SPE returning the results from all ten executions. Tests were done for a variety of values for TASK_SIZE.

![Diagram](image)

**Figure 3.** Diagrammatic overview of the test program for (a) single PPE thread operation, and (b) overall program operation. One PPE thread spawns multiple PPE threads, which make a few decisions before competing for available SPEs to calculate on.

### 3. Results and discussion

Figures 4(a) and 4(b) show the results obtained from execution of the test framework. Tests were performed varying the number of activated SPEs and the number of times each function was looped per PPE thread request. An x86 baseline is shown on the graphs for comparison; this datum indicates the rate
at which a desktop computer can execute the same function. The increase in speed due to the addition of a parallel processing element (i.e. SPE) can be seen as the slope of the curve for a given TASK_SIZE, and is known as the parallelizability. The higher this slope, the greater the benefit of adding more parallel elements; a highly parallelizable algorithm exhibits linear behavior when examined in this way. In Figs. 4(a) and 4(b), we found that there is a significant dependence of acceleration on TASK_SIZE. This is due to the relatively large cost of communication latency between PPE and SPE: a significant amount of time is wasted initializing transfers of data to and from the SPE. As the TASK_SIZE increases, the relative cost of latency decreases (processing and data transfer time increase) and an increase in execution speed is observed (this is most clearly seen when only one SPE is activated). Second, there is a dependency of parallelizability on TASK_SIZE. This is due to the overhead of executing the PPE threads to dispatch requests for function execution on the SPEs. As the TASK_SIZE increases, parallelizability increases because the number of PPE requests for a given number of executions of the test functions decreases. For HOWNEAR, the test framework is limited by the processing power of the PPE at TASK_SIZEs of 1 and 10; the PPE cannot dispatch tasks quickly enough to keep all SPEs in use. At TASK_SIZE = 100 or greater, the PPE is able to dispatch enough requests to provide the SPEs with work to do. This is evidenced by the positive slope of curve shown in Fig. 4(a). This is also the case with RANMAR_GET, though the function requires more cycles to complete than HOWNEAR and thus requires a smaller TASK_SIZE to become parallelizable. It can be seen that under optimal conditions both HOWNEAR and RANMAR_GET achieved acceleration of approximately one half order of magnitude. However, the functions have to be executed in relatively large batches to avoid overloading the PPE and avoiding penalties due to bus transfer latency.
4. Conclusions

We concluded that the algorithms examined in the EGSnrc code are parallelizable on the Cell platform. To implement this, we suggested changing the architecture of the EGSnrc such that EGSnrc is initialized on the PPE, and execution of the particle simulation loop within the program be split between PPE and SPE. The SPE would only perform sections of the code that are appropriate. To sidestep the problem of the tasks being too small (as seen in the test framework for TASK_SIZE < 10), SPE operations would be aggregated before being dispatched to an SPE for calculation. To implement this, we suggest pursuing the architecture that instantiates many simultaneous particle simulations (since each particle is statistically independent) and aggregates small operations across threads for calculation on the SPEs. It is concluded that EGSnrc performance is currently limited on the PS3 by the PPE. For future work, a platform involving a normal PC coupled with a GPGPU may provide a more viable avenue for acceleration. This is due to three factors. First, PC CPUs are much more powerful than the Cell PPE. EGSnrc contains a significant amount of framework code surrounding the actual processing. This is due to three factors. First, PC CPUs are much more powerful than the Cell PPE. EGSnrc contains a significant amount of framework code surrounding the actual processing. This may become a bottleneck if many concurrent particle simulations must be maintained at once - on the PS3, the PPE may be too slow to provide work for the SPEs to do. Second, PCs generally have a much larger available memory space than the PS3s 256MB; currently, it is not uncommon for workstations to have more than 8 GB of available memory. Under the proposed architecture of maintaining multiple simultaneous particle simulations, limited memory may quickly become a performance bottleneck. Finally, CPU and GPGPU technology are being improved much more intensively than the Cell architecture. Hence, gains made by reengineering the EGSnrc for the GPGPU architecture would benefit more from technology upgrades than from reengineering made for the Cell.

Figure 4. Processing speed vs. number of SPEs for subroutine of (a) HOWNEAR, and (b) RANMAR_GET.
References

[1] Fabritiis G De. 2007 Performance of the cell processor for biomolecular simulations, Computer Physics Communications 176, 660-664.

[2] Kurzak J, Buttari A and Dongarra J. 2008 Solving systems of linear equations on the cell processor using Cholesky factorization, IEEE Transactions on Parallel and Distributed Systems 19(9), 1175-1186.

[3] Williams S, Shalf J, Oliker L, Husbands P and Yelick K. 2006 The potential of the cell processor for scientific computing, in: Proceedings of the 3rd conference on Computing frontiers, pp. 9-20.

[4] Williams S, Shalf J, Oliker L, Kamil S, Husbands P and Yelick K. 2007 Scientific computing kernels on the cell processor, International Journal of Parallel Programming 35(3) 263-298.

[5] Kawrakow I and Rogers D W O 2000 The EGSnrc code system: Monte Carlo simulation of electron and photon transport, NRCC Report PIPRS-701, National Research Council of Canada, Ottawa.

[6] Nelson W R, Hirayama H, and Rogers D W O. 1965 The EGS4 code system, SLAC Report 265, Stanford Linear Accelerator Center, Stanford, CA.

[7] Chibani O and Li X A. 2002 Monte Carlo dose calculations in homogeneous media and at interfaces: A comparison between GEPTS, EGSnrc, MCNP, and measurements, Medical Physics 29(5), 835-847.

[8] Chow J C L and Grigorov G N. 2008 Monte Carlo simulation of backscatter form lead for clinical electron beams using EGSnrc, Medical Physics 35(4), 1241-1250.

[9] Verhaegen F. 2002 Evaluation of the EGSnrc Monte Carlo code for interface dosimetry near high-Z media exposed to kilovolt and 60C photons, Physics in Medicine and Biology 47, 1691-1705.

[10] Wang R and Li X A. 2001 Monte Carlo dose calculations of beta-emitting sources for intravascular brachytherapy: A comparison between EG54, EGSnrc and MCNP, Medical Physics 28(2) 134-141.

[11] Rogers D W O. 2006 Fifty years of Monte Carlo simulations for medical physics, Physics in Medicine and Biology 51, R287-R301.

[12] Buttari A, Luszczek P, Kurzak J, Dongarra J and Bosilca G. 2007 A rough guide to scientific computing on the Playstation 3, Technical Report UT-CS-07-595.

[13] Uhrig J L. 1978 System requirements specification for real-time systems, in: Computer software and applications conference, 241-245.

[14] Amdahl G. 1967 Validity of the single processor approach to achieving large-scale computing capabilities, AFIPS conference proceedings 30, 483-485.