Transactions of the Korean Nuclear Society Autumn Meeting
Gyeongju, Korea, October 29-30, 2009

Initial Assessment of Parallelization of Monte Carlo Calculation using Graphics Processing Units

Sung-hoon Choi and Han Gyu Joo
Seoul National University, San 56-1,Ssillim-dong, Seoul, 151-744
hooni86@snu.ac.kr

1. Introduction

Monte Carlo (MC) simulation is an effective tool for calculating neutron transports in complex geometry. However, because Monte Carlo simulates each neutron behavior one by one, it takes a very long computing time if enough neutrons are used for high precision of calculation. Accordingly, methods that reduce the computing time are required. In a Monte Carlo code, parallel calculation is well-suited since it simulates the behavior of each neutron independently and thus parallel computation is natural. The parallelization of the Monte Carlo codes, however, was done using multi CPUs.

By the global demand for high quality 3D graphics, the Graphics Processing Unit (GPU) has developed into a highly parallel, multi-core processor. This parallel processing capability of GPUs can be available to engineering computing once a suitable interface is provided. Recently, NVIDIA introduced CUDA™, a general purpose parallel computing architecture.[1] CUDA is a software environment that allows developers to manage GPU using C/C++ or other languages. In this work, a GPU-based Monte Carlo is developed and the initial assessment of it parallel performance is investigated.

2. Methods

A simplified mutigroup 2-D MC code was written first for this development. The CPU based code performs the I/O processing as well as the transport simulation. Only the transport simulation part is made to be executed on GPU.

In the parallel implementation, errors in tally could occur if several threads access and changes a variable stored in the same memory simultaneously. To avoid this, a variable storing a tally should be given as an array that has the number of threads elements. By the same reason, the variables storing new neutrons generated by fission are also made thread dependent. After all the threads complete the transport simulation, the new fission neutron data stored separately are unified in one queue and distributed to the threads at the next cycle.

The algorithms that check and manage how many threads are active in the current cycle are added in the parallel implementation. In the loop in which neutron data are distributed to each thread, the loop terminates i) if the neutrons are launched to all the threads, or ii) if the total number of simulated neutrons in the current cycle equals to the number of neutrons generated by fission in the previous cycle. For example, if there are 1500 threads and 10000 new fission neutrons, all 1500 threads are active during 1st - 6th simulations, but because there are only 1000 neutrons left, 1000 threads are active and rest 500 are inactive at 7th simulation.

The random number generator (RNG) is also a problem in parallelization. If the same RNG are used with the same seed, every thread will perform exactly the same simulation. Thus, the parallel RNG are required for parallelization. A parallel RNG available from the CUDAMCML, Monte Carlo code for photon transport developed Lund University is used here.[3]

3. Performance Assessment

The CPU model used to compare performance is Intel Core2 Quad Q9400 2.66GHz while the follow three GPUs models are used: GeForce 8400M GS, GeForce 9600 GT, and Tesla C1060. They have 2, 8 and 30 multi-processors (MP), respectively and each MP contains 8 processors.

2.1. Simulation condition

The test problem used for this test is the C5G7MOX core problem. The various numbers of source neutrons per cycle ranging from 250 to 1000000 are tried and the total number of cycles is 100 with the number of inactive cycles of 20.

In order to maximize the GPU computing potential, the number of threads should be selected properly. With more threads, more parallel simulations are done on GPUs. However, if there are too many threads, the number of registers per thread can be insufficient and then GPU processing speed gets lower.

In this work, the numbers of threads are selected with the execution data for the case of 10000 neutrons per cycle. The optimum numbers of threads are selected such that the simulation time becomes the minimum. The resulting numbers of threads are 3200, 8000, and 12000, respectively, for GeForce 8400M GS, GeForce 9600 GT, and Tesla C1060. In fact, if the number of threads is selected by calculating number of needed resisters, the performance of GPU would improve.

2.2. Comparison accuracies of CPU and GPU

The k-eff's obtained with various numbers of source neutrons on different platforms are show in Figure 1 with the error bar designating 1σ standard derivation. It is shown in this figure that all k-eff's of CPU and GPUs agree within the 1σ range. Therefore, the results of CPU

and GPUs can be regarded the same in the statistical point of view.

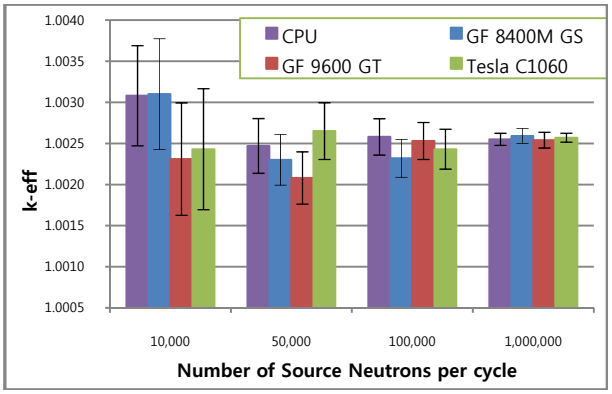

Fig1.The results of calculating k-eff on CPU and GPU

2.3. Comparison performances of CPU and GPU

Table 1 lists the computing time for the various cases and Figure 2 show the speedup.

Table 1. Monte Carlo simulation time with CPU and GPU

No. of Neutron per cycle	CPU [sec]	GeForce 8400M GS [sec]	GeForce 9600 GT [sec]	Tesla C1060 [sec]
250	1.09	1.80	1.84	1.09
500	1.79	2.22	1.86	1.16
750	2.54	2.47	1.89	1.19
1,000	3.20	2.75	1.92	1.22
3,000	9.32	5.35	2.42	1.45
5,000	14.40	8.78	2.90	1.63
7,000	20.15	11.77	3.37	1.86
10,000	29.93	15.86	4.59	2.14
50,000	147.97	73.39	16.93	8.48
100,000	298.18	145.08	32.17	15.92
1,000,000	2839.35	1406.23	306.13	144.02

It is shown in this table that GPUs are not any faster than CPU if the number of neutrons per cycle is too few. But for the practical cases in which more neutrons are used for low variance in the local pin power parameters, there are substantial speedups by using GPUs. The speedup for GeForce 8400M GS, GeForce 9600 GT, and Tesla C1060 are about 2, 10 and 20, respectively.

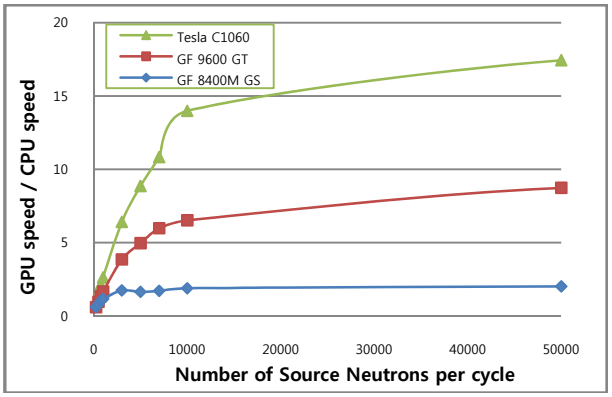

Fig 2. Comparison of GPU speedups

If the number of neutrons per cycle is low, not all threads are operating. Instead, the time of copying data between GPU and CPU takes a lot of overhead and thus GPU is slower than CPU. But if the number of neutrons per cycle is high enough, all threads operate and the simulation is accelerated effectively by parallel computing.

4. Conclusion

A simple multigroup, 2-D GPU-based Monte Carlo was developed to examine its parallel execution performance. With a large number of neutrons per cycle, the simulation speed could be about 20 times faster than the CPU-based one on the Tesla C1060 machine. Although a GPU core can't give the performance similar to a CPU core, the fact that there are considerable many cores (240 cores in the case of Tesla C1060) provides much room for accelerating through improving the parallel algorithm. This research suggests that with much less price and smaller space, GPUs can deliver better performance than CPU clusters in parallel Monte Carlo calculations.

REFERENCES

[1] "NVIDIA CUDA™ Programming Guide", Ver. 2.2, pp. 1-3, 2009.
[2] P. Martinsen, J. Blaschke, R. Kunnemeyer, and R. Jordan, "Accelerating Monte Carlo simulations with an NVIDIA graphics processor", ELSEVIER, pp. 1-7, 2009.
[3] E. Alerstam, T. Svensson, and S. Andersson-Engels, "CUDAMCML User manual and implementation notes", Department of Physics Lund University, pp. 12, 2009.