The State of Monte Carlo Neutron Transport: The Role of GPUs and Portable Performance Abstractions

Ryan Bleile
School of Computer and Information Science
University of Oregon
Eugene, Oregon 97403
Email: rbleile@cs.uoregon.edu

***Abstract*—Since near the beginning of electronic computing Monte Carlo neutron transport has been a fundamental approach to solving nuclear physics problems. Over the past few decades Monte Carlo transport applications have seen significant increase in their capabilities and decreases in time to solution. Research efforts have been focused on areas such as MPI scalability, load balance with domain decomposition, and variance reduction techniques. In the last few years however the landscape has been changing. Due to the inherently parallel nature of these applications Monte Carlo transport applications are often used in the super-computing environment. Supercomputers are changing, becoming increasingly more parallel on node with GPGPUs and/or Xeon Phi co-processors powering the bulk of the compute capabilities. In order to fully utilize the new machines capabilities it is becoming ever more important to migrate to a many-core perspective of any computing algorithms. Monte Carlo transport applications, like many others, have the potentially difficult task of figuring out how to effectively utilize this new hardware. Many groups have taken the initial steps to look into this problem or have focused their efforts on a sub-problem, such as continuous energy lookups. In other fields a promising approach is displayed in the use of portable performance abstractions in order to specify what is parallel. These abstractions provide the foundations for applications to write code once and run on any supported platform. This paper describes the state of the art in Monte Carlo neutron transport, with a special emphasis on the affects of upcoming architectures.**

CONTENTS

I. INTRODUCTION

Today's Supercomputer landscape is in flux. Supercomputer architectures are making more extreme changes then they have undergone in 20 years. One big driving factor for this change are the concerns about energy usage as we scale to larger and larger machines. A metric, FLOPS/Watt is often used to describe this relationship. In order to maximize the FLOPS/Watt metric, architectures are shifting away from fast and complex multi-core CPUs and adding in much larger numbers of much slower simpler processors. The amount of parallelism available on any given node in a supercomputer is growing but factors of hundreds or thousands because of this change. This change brings new and interesting challenges that need to be overcome.

In addition to the increase in node level parallelism, it is unclear which architecture choice will prove to be a winning design. Currently there are many different architectures to choose from when designing a supercomputer and there is no obvious choice to place one design above the others, or if any of these designs are going to end up above the others. NVIDIA provides General Purpose Graphics Processing Units (GPGPUs) which are a highly parallel throughput optimized devices. Intel provides their Many Integrated Core (MIC) co-processor which provides large vector lanes and many threads. Other groups are turning to Field Programmable Gate Arrays (FPGAs) for a solution. Across the Department of Energy (DOE) National Labs both the NVIDIA and Intel approaches are being pursued in their newest procurements [1], [2].

Application developers now face a complex and unclear path forward. There is additional levels of complexity and potentially large application changes that will need to be made in order to effectively utilize this increase in parallelism. In addition, an application programmer cannot simply begin a cycle of porting to a new hardware architecture. Instead, applications need to address the issue of portability as well as performance or they run the risk of becoming outdated or unusable very quickly. This problem is especially challenging when optimization choices for one architecture can contradict with optimization choices on another architecture.

There are a large number of physics and multi-physics applications that exist today that must figure out how to navigate this complex and challenging landscape. Simple ports to new architectures are often not enough to guarantee performance, and will still require applications to be ported multiple times to multiple architectures. This paper will explore these concerns and today's current efforts for portable performance solutions in the scope of one of these physics applications, Monte Carlo Neutron Transport.

II. What is Monte Carlo Neutron Transport?

> ”The first thoughts and attempts I made to practice [the Monte Carlo method] were suggested by a question which occurred to me in 1946 as I was convalescing from an illness and playing solitaire. The question was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than ”abstract thinking” might not be to lay it out say one hundred times and simply observe and count the number of successful plays. This was already possible to envisage with the beginning of the new era of fast computers, and I immediately thought of problems of neutron diffusion and other questions of mathematical physics, and more generally how to change processes described by certain differential equations into an equivalent form interpretable as a succession of random operations. Later... [in 1946, I] described the idea to John von Neumann and we began to plan actual calculations.”
>
> - Stan Ulam 1983 [3]

John von Neumann became interested in Stan Ulams idea and was outlined in detail how to solve the neutron diffusion and multiplication problems in fission devices. Since this time Monte Carlo methods have continued to be a primary way for solving many questions in Neutron transport. [3]

A. Definition

E. E. Lewis and W. F. Miller Jr. in Computational Methods of Neuron Transport describes Monte Carlo transport as a simulation of some number of particle histories by using a random number generator. For each particle history that is calculated, random numbers are generated and used to sample from probability distributions describing the different physical events a particle can undergo, such as scattering angles or the length between collisions. Ivan Lux and Laslo Koblinger further this definition in their book Monte Carlo Particle Transport Methods: Neutron and Photon Calculations:

> ”In all applications of the Monte Carlo method a stochastic model is constructed in which the expected value of a certain random variable (or of a combination of several variables) is equivalent to the value of a physical quantity to be determined. This expectation value is then estimated by the averaging of several independent samples representing the random variable introduced above. For the construction of the series of independent samples, random numbers following the distributions of the variable to be estimated are used.” [4]

Estimating this type of quantity takes on this mathmatical form:

$$\hat{x} = \frac{1}{N}\sum_{n=1}^{N} x_n,$$

where x_n represents the contribution of the nth history for that quantity. And so for the Monte Carlo method we tally the x_n from each particle history in order to compute the expected value $\hat{x}$. [5]

One very important question to answer with any statistical answer is just how good is our estimated value $\hat{x}$ when we compare to the true value $\bar{x}$. It turns out that the uncertainty in $\hat{x}$ decreases with increasing numbers of particle histories, and generally falls of asymptotically proportionate to $N^{-1/2}$. [5]

B. The Equation

The equation being solved by the neutron transport problem, shown below, displays each of the pieces that makes up a full Monte Carlo transport algorithm. This equation is known as the Linearized Boltzmann transport equation for neutrons:

$$\begin{gathered}
\frac{1}{\nu}\frac{\partial\Psi(\vec{r},E,\Omega,t)}{\partial t} + (\nabla\cdot\Omega)\Psi(\vec{r},E,\Omega,t) + \Sigma_a(\vec{r},E)\Psi(\vec{r},E,\Omega,t) \\
= \\
\int_{E'}\int_{\Omega'}\Sigma_s(\vec{r},E',\Omega' \to E,\Omega)\Psi(\vec{r},E,\Omega,t)d\Omega' dE' \; + \\
\chi(E)\int_{E'}\nu(E')\Sigma_f(\vec{r},E')\int_{\Omega'}\Psi(\vec{r},E,\Omega,t)d\Omega' dE' \; + \\
S_e xt(\vec{r},E,\Omega,t)
\end{gathered}$$

[6] where $\Psi(\vec{r},E,\Omega,t)$ is angular flux, $\Sigma_a(\vec{r},E)$ is the macroscopic cross section for particle absorption, $\Sigma_s(\vec{r},E',\Omega' \to E,\Omega)$ is the macroscopic cross section for particle scattering, $\Sigma_f(\vec{r},E')$ is the macroscopic cross section for particle production from a fission collision source, $\chi(E)$ is a secondary particle spectrum from he fission process, $\nu(E)$ is the average number of particles emitted per fission, $S_e xt(\vec{r},E,\Omega,t)$ represents an external source, $\vec{r}$ is the spacial coordinates, E is the energy, Ω is angular direction, and t is the term for time. [6]

C. Algorithm Approach

While there are many ways people can solve this problem generally people consider tracking particle histories and the algorithm that follows is often called the history based approach because of this. Algorithm 1 shows the history based approach for a simple research code [7] but has the same properties we can use to describe the method. Algorithm 2 shows the outer most scope of a Monte Carlo Problem for referencing where different optimizations or stages occur. For example in Algorithm 1 this takes place inside the Cycle Loop and shows only the steps for Cycle Tracking.

Algorithm 1: History based Monte Carlo algorithm

```
foreach particle history do
    while particle not escaped or absorbed do
        sample distance to collision in material
        sample distance to material interface
        compute distance to cell boundary
        select minimum distance, move particle, and
        perform event
        if particle escaped spatial domain then
            update leakage tally
            end particle history
        if particle absorbed then
            update absorption tally
            end particle history
```

Algorithm 2: Monte Carlo Method

```
Parse Inputs
foreach Cycle do
    Cycle Initialize
    Cycle Tracking
    Cycle Finalize
Gather Tallies
```

III. State of the Art Research

In order to understand the state of the art research in Monte Carlo transport, it is important to have some perspective on what has been done in the field already. There is a long history of research and improvements for Monte Carlo transport problems and understanding this path and the machines that that research was designed for can help guide analysis of more recent efforts. In the field of Monte Carlo transport most research in the last 5 years has been related to GPGPU computing or has been physics based as apposed to computer science based research. We will review and discuss the GPU research in a later section. In this section we will look at the significant areas of interest when dealing with Monte Carlo transport applications that do not pertain to GPUs, namely Parallel performance on different architectures, load balancing, optimizations in nuclear data look-ups, and research dealing with variance reductions.

A. Parallel Performance

In this section we will see the parallel performance of a number of different Monte Carlo particle transport applications on different architectures ranging from the vector machines of the 80's to multi-core compute clusters.

Shared Memory Performance

Shared memory systems refer to machines or models where all processors can access the same memory space. Taking this a step further the unified memory architecture (UMA) shared memory systems not only do all processors have access to the same memory but they also have access to all memory in the same time [8]. One type of shared memory system that was popular in the 1970's and 1980's was the vector machine. Vector machines took the shared memory system and added additional synchronicity to the system by making all of the processors issue the same instruction [9].

Vector Machine Performance

In the 1980's Monte Carlo transport algorithms began adapting "event-based" methods in order to vectorize their algorithms for use on a vector machine. These new algorithms were used because the traditional history based approach has complete independence of particle histories. But in order to effectively utilize the vector architecture particles must be computed on the same code paths. By changing the algorithm to follow events instead of histories the Monte Carlo method could be used in a vector based approach. [10]

One common element when reviewing the work done in this area is to see that the vector approach is often related to stacks, and properly organizing particles into the right stack so that calculations can be preformed [11], [12]. Another approach is to try to use only one main stack and pull off only the minimum information needed to compute the events into sub-stacks [10]. With each of these approaches particle events determine how the particles are organized or what information is needed for processing. The main drawback to the event based approach is the added time processing data movement or sorting.

Brown reported theoretical speedups of 20x-85x for his consideration and deemed this approach well worth the efforts required to change codes around in order to use this approach [11]. Martin saw speedups randing from 5x to 12x depending on the problem and the machine he was running on by using the single big stack, sub-stack approach [10]. Bobrowicz explicit stack approach reaches speedups of around 8x - 10x compared with the original history-based approach [12]. Finally Burns in using a LANL Benchmark code GAMTEB showed he could acheive similar performance to Bobrowics by following a similar approached as that laid out by Brown [13].

TABLE I
PARALLEL PERFORMANCE ON THE MTA USING MULTITHREAING

Procs	Time (sec)	Speedup	Efficiency
Parallelization by zones only			
1	764	1.00	1.00
2	400	1.91	0.95
4	227	3.37	0.84
8	167	4.58	0.57
Parallelization by zones and energies			
1	745	1.00	1.00
2	370	2.01	1.01
4	187	3.98	0.99
8	94	7.92	0.99

Multi-Threaded Architecture Performance

Other shared memory systems, separate from vector machines, where tried in this time. One such machine was the Tera Multi-Threaded Architecture (MTA). This approach focused on the use of incredibly parallel processors, hardware threading, and a simple shared memory, no cache, design. The idea was by focusing on threading they could mask away memory latency [14], [15].

One Photon transport application tried two methods of parallelizing their application on the Tera MTA. For their problem the zones and energies of the region needed to be looped over and photons falling in those ranges where then computed. So for their application they chose to parallelize over zones and also over zones and energies at once through loop unrolling. Table I shows that the parallelization on the MTA over zones and energies maintains incredible efficiency giving their application good speedups here, while parallelizing over only zones does not expose enough parallel work to hide memory latency and so efficiency drops of quickly.

More modern systems utilize shared memory ideas as well, with a majority of the scientific efforts utilizing openMP threading models for shared memory processing. Often this model is overlooked in preference of distributed computing via MPI but that is not always the case. Given an all particle method, openMP codes tend to scale incredibly well with the only draw backs having to do with those few areas requiring atomic operations. With, in my experience, an nearly perfect efficiency on a node, in the case of no atomic operations and plenty of work.

Distributed Memory Performance

One of the major transitions in supercomputing came with the shift from vector computing to distributed memory computing. This type of computing is most often done with MPI and has, for the last 20 years, been a primary method of achieving parallel performance on clusters and supercomputers alike. In the message passing model of parallelism, independent processes work together through the use of messages to pass data between processors. In this model, parallel efficiency is generally improved by spending more time working independently and is negatively affected by time spent sending messages [16].

The Monte Carlo particle transport history-based approach lends itself to the distributed model very well. Since each particle history is independent of any other particle histories and can be easily split up over processors [16]. The only complications we normally see when we move to the distributed memory systems is that we often use domain decomposition which increases the complexity and use of the message passing interface. We will discuss the domain decomposition challenges in the load balance section under state of the art research.

Given the embarrassingly parallel nature of the Monte Carlo transport problem, the performance of this model produces results as suspected. As we increase MPI processes we continue to get a nearly liner speedup. Majumdar shows that with 16 nodes and 8 MPI tasks per node, his biggest run, he was still able to achieve a 88% efficiency [14] in his code that was parallel over zones and energies. In an all particle code, Mercury at LLNL, we can see parallel efficiencies of 80-90% when using MPI parallelism [17].

Distributed + Shared Memory Performance

Given the heterogenous nature of todays computing environment, and even in the fairly homogenous environment we are leaving, it is a common next step to consider combining distributed and shared memory parallel schemes. It seems an obvious extension to either of these models to add the other. Shared memory parallelism exists on a node or on one of the new accelerator devices. Distributed memory parallelism provides the opportunity for scaling to large supercomputers or clusters, giving you many nodes to work with. Given the nature of these two models it is surprising how often the additional extension of combining them is not done as most often shared memory models are overlooked in favor of distributed models, and since the distributed model works ”well enough” even with in a node, it is often not worth the effort to try to combine these two methods. This is no longer going to be true when we start adding accelerators and many people have found benefits of combining these models anyways.

The combined distributed-shared model is often referred to as MPI+X [18]. The X in this description being replaced with whichever shared memory system is preferred. The most common implementation of MPI+X to date is the MPI + OpenMP model. Utilizing MPI for node to node communication and OpenMP for on node parallelism [18].

Yang has recently shown that the MPI+OpenMP model has benefits of achieving the nearly perfect parallel efficiency one would expect as well as significantly decreasing the memory overhead to an equivalent MPI only implementation. He was able to show 82-84% parallel efficiencies and a decrease in memory cost from 1̃.4GB to 2̃00MB for 8 processors [16].

Majumdar shows that with 16 nodes and 8 OpenMP threads per node, he was able to achieve a 95% parallel efficiency which is an improvement over the his MPI only methods 88% parallel efficiency [14].

B. Load Balance and Domain Decomposition Algorithms

A look at load balance algorithm and MC at scale.

[19] [20] [21] [22] [17] [23]

C. Nuclear Data

A look at the areas of nuclear data look-ups and improvements happening in the non GPU world.

D. Variance Reduction Techniques

current state of variance reduction research

IV. State of the Arr: GPU Research

a look at gpu research in MC

A. First Pass at GPU Computing

Research papers on peoples fists attempts at GPU computing with varying results.

B. Monte Carlo and Medicine

A look at the GPU research being done in the medical field of monte carlo transport

C. Monte Carlo and Ray Tracking

a look at the combined efforts to put ray tracing techniques into monte carlo applications. MC and optix.

D. Event Based Techniques

A look at old and new event based approaches for mc applications.

V. What is Portable Performance

The term portable performance generally means the ability to achieve a high level of performance on a variety of architectures. In this case high performance is relative to each target system [24]. One important consideration then is what variety of systems are used that applications need to be portable for.

The top ranked machines in the world currently utilize technologies like general purpose graphics processing units (GPUs, e.g., NVIDIA Tesla in Titan), many-core coprocessors (e.g., Intel Xeon Phi in Tianhe-2), and large multicore CPUs (e.g., IBM Power, Intel Xeon in Tianhe-2 and others) [24], [25], [26]. Further, future supercomputing designs may include low-power architectures (e.g., ARM), hybrid designs (e.g., AMD APU), or experimental designs (e.g., FPGA systems) [26]. Given this wide array of possible architectures the value of portable performance has never before been so high.

VI. Monte Carlo and Portable Performance

a look at monte carlo and pp abstractions. My alps work

References

[1] "Coral/sierra." [Online]. Available: https://asc.llnl.gov/coral-info

[2] "About trinity." [Online]. Available: http://www.lanl.gov/projects/trinity/about.php

[3] R. Eckhardt, "Stan ulam, john von neumann, and the monte carlo method," *Los Alamos Science*, vol. 15, no. 131-136, p. 30, 1987.

[4] I. Lux and L. Koblinger, *Monte Carlo Particle Transport Methods: Neutron and Photon Calculations*. 2000 Corporate Blvd., Boca Raton, Florida 33431: CRC Press, Inc, 1991.

[5] E. E. Lewis and J. W. F. Miller, *Computational Methods Of Neutron Transport*. 555 N. Kensington Avenue La Grange Park, Illinois 60525 USA: American Nuclear Society, Inc., 1993.

[6] N. Gentile, R. Procassini, and H. Scott, "Monte carlo particle transport: Algorithm and performance overview," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2005.

[7] R. Bleile, P. Brantley, S. Dawson, M. O'Brien, and H. Childs, "Investigation of portable event-based monte carlo transport using the nvidia thrust library," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2016.

[8] H. El-Rewini and M. Abd-El-Barr, *Advanced computer architecture and parallel processing*. John Wiley & Sons, 2005, vol. 42.

[9] R. M. Russell, "The cray-1 computer system," *Communications of the ACM*, vol. 21, no. 1, pp. 63–72, 1978.

[10] W. R. Martin, P. F. Nowak, and J. A. Rathkopf, "Monte carlo photon transport on a vector supercomputer," *IBM Journal of Research and Development*, vol. 30, no. 2, pp. 193–202, 1986.

[11] F. B. Brown and W. R. Martin, "Monte carlo methods for radiation transport analysis on vector computers," *Progress in Nuclear Energy*, vol. 14, no. 3, pp. 269–299, 1984.

[12] F. Bobrowicz, J. Lynch, K. Fisher, and J. Tabor, "Vectorized monte carlo photon transport," *Parallel Computing*, vol. 1, no. 3, pp. 295–305, 1984.

[13] P. J. Burns, M. Christon, R. Schweitzer, O. M. Lubeck, and H. J. Wasserman, "Vectorization on monte carlo particle transport: an architectural study using the lanl benchmark gamteb," in *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*. ACM, 1989, pp. 10–20.

[14] A. Majumdar, "Parallel performance study of monte carlo photon transport code on shared-, distributed-, and distributed-shared-memory architectures," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*. IEEE, 2000, pp. 93–99.

[15] A. Snavely, L. Carter, J. Boisseau, A. Majumdar, K. S. Gatlin, N. Mitchell, J. Feo, and B. Koblenz, "Multi-processor performance on the tera mta," in *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 1998, pp. 1–8.

[16] F. YANG, G. YU, and K. WANG, "Hybrid shared memroy/message passing parallel algorithm in reactor monte carlo code rmc," in *Proceedings of the Reactor Physics Asia 2015 Conference*, 2015, pp. 16–18.

[17] R. Procassini, M. O?Brien, and J. Taylor, "Load balancing of parallel monte carlo transport calculations," *Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications, Palais des Papes, Avignon, Fra*, 2005.

[18] M. Wolfe, "Compilers and more: Mpi+x," HPC wire, jul 2014, https://www.hpcwire.com/2014/07/16/compilers-mpix/.

[19] M. O?Brien and P. Brantley, "Particle communication and domain neighbor coupling: Scalable domain decomposed algorithms for monte carlo particle transport," *Lawrence Livermore National Laboratory (LLNL), Livermore*, 2015.

[20] M. J. O'Brien, P. S. Brantley, and K. I. Joy, "Scalable load balancing for massively parallel distributed monte carlo particle transport," in *Proceedings of International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013), Sun Valley, Idaho*, 2013.

[21] M. O'Brien, K. Joy, R. Procassini, and G. Greenman, "Domain decomposition of a constructive solid geometry monte carlo transport code," in *Int. Conf. Adv. Math., Comput. Methods, Reactor Phys*, 2009.

[22] G. Greenman, M. O'Brien, R. Procassini, and K. Joy, "Enhancements to the combinatorial geometry particle tracker in the mercury monte carlo transport code: Embedded meshes and domain decomposition," in *International conference on mathematics, computational methods and reactor physics*, 2009.

[23] M. O'Brien, J. Taylor, and R. Procassini, "Dynamic load balancing of parallel monte carlo transport calculations," *The Monte Carlo Method: Versatility Unbounded In A Dynamic Computing World*, pp. 17–21, 2005.

[24] M. Wolfe, “Compilers and more: What makes performance portable?” HPC wire, apr 2016, https://www.hpcwire.com/2016/04/19/compilers-makes-performance-portable/.

[25] “June 2016,” Top 500 The List, jun 2016, https://www.top500.org/lists/2016/06/.

[26] H. Childs, “Portable performance,” 2015, http://cdux.cs.uoregon.edu/portable_perf.html.