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:

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

[6] where $\Psi(\vec{r},E,\Omega,t)$ is angular flux, $\Sigma_a(\vec{r},E)$ is the macroscopic cross section for particle absorption, $\Sigma_s(\vec{r},E',\Omega'\rightarrow E,\Omega)$ 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 4 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 4 this takes place inside the Cycle Loop and shows only the steps for Cycle Tracking.

Algorithm 1: History based Monte Carlo algorithm

```
1  foreach particle history do
2      while particle not escaped or absorbed do
3          sample distance to collision in material
4          sample distance to material interface
5          compute distance to cell boundary
6          select minimum distance, move particle, and
           perform event
7          if particle escaped spatial domain then
8              update leakage tally
9              end particle history
10         if particle absorbed then
11             update absorption tally
12             end particle history
```

Algorithm 2: Monte Carlo Method

```
1 Parse Inputs
2 foreach Cycle do
3     Cycle Initialize
4     Cycle Tracking
5     Cycle Finalize
6 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. There has been a tremendous growth in the Monte Carlo industry since its inception over 60 years ago. The first models developed in run in 1947 would take five hours to compute 100 collisions, a tasks that today can be done in milliseconds. In the 1940's and 1950's Monte Carlo codes were written in very low level languages on the earliest computers. The 1960's to 1980's saw a great increase in the capabilities of the Monte Carlo codes. In the 1980's Monte Carlo codes adopted vector machines and parallel/vector computers. In the 1990's Monte Carlo become more common place and parallelism increased to 100s or 1000s or processors through PVM or MPI. In the 2000's multicore processors meant threading become more common place mixing local and global forms of parallelism reaching 10,000s or processors. [8]

This growth in computer processing can also be categorized in terms of the styles of memory accesses. Early systems were shared memory environments almost exclusively. Then distributed memory systems become popular and finally the combination of distributed and shared memory systems became popular.

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 [9]. 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 [10].

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. [11]

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 [12], [13]. 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 [11]. 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 [12]. 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 [11]. Bobrowicz explicit stack approach reaches speedups of around 8x - 10x compared with the original history-based approach [13]. 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 [14].

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 [15], [16].

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.

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

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 [17].

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 [17]. 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 [15] 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 [18].

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 [19]. 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 [19].

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 $\tilde{1}$.4GB to $\tilde{2}$00MB for 8 processors [17]. 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 [15].

B. *Load Balance and Domain Decomposition*

In order to achieve high levels of parallelism in transport problems with many geometries or zones different parallel execution models are used. Two primary models used are domain decomposition and replication. Domain decomposition involves spatial decomposition of the geometry into domains, and then the assigning of processors to work on specific domains. Replication involves storing the geometry information redundantly on each processor and assigning each processor a different set of particles. [20] [18]

Load balance is often discussed in conjunction with domain decomposition since particles often migrate between different regions of a problem and so not all spatial domains will require the same amount of computational work. In many applications there is at least one portion of the calculation that must be completed by all processors before all the processors can move forward with the calculation. If one processor has more work than any other all of the others must wait for that processor to complete its work [20] [18]. This load-imbalance can cause significant issues with scalability as paralleism is increased from hundreds to millions of processors [21].

When to Load balance

One key consideration when wanting to preform a load balance calculation is to understand the cost of preforming that calculation as well. If too much time is spent making sure the problem is always perfectly load balanced then computational resources are being wasted on a non-essential calculation resulting is overall slower performance. However, if too little resources are devoted to load-balancing than the problem will suffer from load-imbalance and the negative effects that entails. One solution is to preform load balance at the start of each cycle or iteration of a Monte Carlo transport calculation but only when that load balance will result in a faster overall calculation [20] [18].

An algorithm to determine when to load balance was explained in references [20] [18] where the following criterion could be checked inexpensively each cycle to determine if a load-balance operation should take place. First, is to compute a speedup factor by comparing current parallel efficiency (ε_C) to what parallel efficiency would be if processors were to redistribute their load (ε_{LB}). Second, is to predict the run time by using the time to execute the previous cycle(τ_{Phys}), the speedup factor(S), and finally the time to compute the load balance itself(τ_{LB}). Finally ,is to compare the predicted run with and without load balancing to determine if the operation is worth while. [20] [18]

$$S = \frac{\varepsilon_C}{\varepsilon_{LB}} \tag{1}$$

$$\tau^{'} = \tau_{Phys} \cdot S + \tau_{LB} \tag{2}$$

$$\tau = \tau_{Phys} \tag{3}$$

$$if\ (\tau^{'} < 0.9 \cdot \tau\ )\ DynamicLoadBalance(); \tag{4}$$

Extended Domain Decomposition

As an extension to the domain decomposition of meshes, O'Brien and Joy demonstrated an algorithm to domain decompose Constructive Solid Geometry (CSG) in a Monte Carlo transport code. One key difference between mesh and CSG geometries is that mesh geometries contain a description of cell connectivity where as cells defined though CSG do not. In order to domain decompose these CSG cells each cell was given a bounding box and since each domain is also a box a test for if a cell belongs inside a domain becomes an axis-aligned box-box intersection test. [22]

In addition to pure mesh and pure CSG problems other combinations might be useful, such as the combination of mesh and CSG problems where there are large-scale heterogeneous and homogeneous regions. In this method a mesh region is embedded inside a CSG region allowing for the use of each in whichever region one or the other is more optimal. [23]

Load Balance at Scale

When load balancing massively parallel computers it is unscalable to need to examine the workload of every processor. O'Brien, Brantley and Joy present their scalable load balancing algorithm that runs in $\Theta(log(N))$ by using iterative processor-pair-wise balancing steps that will ultimitely lead to a balanced workload. Their algorithm shows remarkable ability to load balance and is demonstrated up to 2 million processors on the Sequoia supercomputer at Lawrence Livermore National Laboratory. [21]

The pair-wise load balancing scheme maintained a high efficiency; with the load-balanced runs maintaining efficiencies of 95% at 2 million processors when the not load-balanced runs continuously drops to around 68% efficiency at 2 million

processors. In addition the load-balanced version is able to maintain near perfect scaling up to 2 million processors. By dispersing the workload over processors effectively it also decreases the overall tracking time. [21]

Algorithms that interact with the particles and geometries can also be revisited after domain decomposition is added. Specifically a Global Particle Find algorithm, a test for done, and domain neighbor replication. The global particle find can be solved with a simple tree search; even though building the tree is not a scalable algorithm it is fast. By using MPI_I allreduce() for the global test for done in place of a complex hand coded algorithm scalability and ease of maintenance is achieved. Lastly, the domain neighbor replication algorithm ended up being very important or scalability and achieved 100% load balance and reduced memory usage by using a recursive Euclidean GCD algorithm to build a bipartite graph between adjacent domains. [24]

C. Nuclear Data

A large part of many Monte Carlo transport calculations is the process of looking up nuclear data information. Both microscopic and macroscopic cross section information is needed in order to understand what reactions a particle undergoing a collision will do. Depending on the problem and the choices made to solve it, time spent looking up nuclear data can often be between 10% and 85% of the over all runtime. The problems that spend more time looking up cross section data are often using what is known as the continuous energy model where energy value are stored as a large sequence of points and exact values are found through interpolation. The second method that is used which makes cross section lookups faster but less accurate is multi-group cross sections, where cross section data is stores in some number of bins and all energies that land in the bin are given the same value. This can often reduce the search many orders of magnitude.

Research that deals with nuclear data lookups is often concerned with speeding up the search for a given cross section at a given energy. This search problem is the main bottleneck in the cross section lookup algorithms. Linear searches, binary searches, and Hash based searches are often employed for this. In addition combining isotopes into a unionized grid is a common method for reducing the total number of searched required, though it greatly increases the memory needed to store the cross section data.

Each of the common competing continuous algorithms is well defined and compared by Wang et. al. and are described as follows [25]:

Hashing: : Each material's whole energy range is divided up into N equal intervals, and for every individual isotope inside the material an extra table is established to store isotopic bounding indexes of each interval [26]. The new search intervals are thus largely narrowed with respect to the original range and can be reached by a single float division. The hashing can be performed on a linear or logarithmic scale; the search inside each interval can be performed by a binary search or linear search. In the original paper [26], a logarithmic hashing was chosen with $N \simeq 8000$ as the best compramize between performance and memory usage. Another variant is to perform the hashing at the isotope level.

Unionized grid: : A global unionized table gathers all possible energy points in the simulation and seconds table provides their corresponding indexes in each isotope energy grid [27]. Every time an energy lookup is performed, only one search is required in the unionized gris and the isotope index are directly provided by the secondary index table. Timing resutls show that this method has a significant speedup over the conventional binary search but can require up to a $36\times$ more memory space [28].

Fractional cascading: : This is a technique to speedup search operations for the same value in a series of related data sets [28]. The basic idea is to build a unified grid for the first and second isotopes, then for seconds and third, etc. When using the mapping technique, once we find the energy index in the first energy grid all the following indexes can be read directly from the extra index teables without further computations. Compared to the global unionized methods, the fractional cascading technique greatly reduces memory usage.

We will discuss recent work done in the area of nuclear data lookups when we discuss many-core based Monte Carlo research as much of the results are targeted at NVIDA GPGPUs or the Intel XEON Phi many-core coprocessor.

D. Variance Reduction Techniques

Variance reduction is a key concept in Monte Carlo transport problems. Often without some use of variance reduction certain problems would take an incredible amount of time and computing power to begin finding a solution. The idea behind variance reduction is to increase the efficiency of Monte Carlo calculations and permit the the reduction of the sample size in order to achieve a fixed level of accuracy or increase accuracy at a fixed sample size [29]. Some commonly used variance reduction techniques are common random numbers, antithetic variates, control variates, importance sampling and stratified sampling, although most used in Monte Carlo transport is some form of importance sampling.

Common Random Numbers:: This method of variance reduction involves comparing two or more alternative configurations instead of only a single configuration. Variance reduction is achieved by introducing an element of positive correlation between the sets. [30]

Antithetic Variates: : This method of variance reduction involves taking the antithetic path for each path sampled — so for a given path $\{\varepsilon_1, ..., \varepsilon_M\}$ one would also take the path $\{-\varepsilon_1, ..., -\varepsilon_M\}$. This method reduces the number of samples needed and reduces the variance of the sampled paths. [31]

Control Variates: : This method of variance reduction involves creating a correlation coefficient by using information about a known quantity to reduce the error in an unknown quantity. This method is equivalent to solving a least squares system and so is often called regression sampling. [32]

Importance Sampling: : This method of variance reduction involves estimating properties of a particular distribution, while only having samples generated from a different distribution than the distribution of interest. This method emphasizes important values by sampling them more frequently and sampling un important values less frequently [33]. This is often achieved through methods known as splitting or Russian roulette. In splitting and Russian roulette particles are each given a weight and if particles enter an area of higher importance they are split into more particles with less weight giving a larger sample size. If particles travel in a region that is not important they undergo Russian roulette where some particles are killed off and others are given a heavier weight to account for those removed. [34]

Stratified Sampling: : This method of variance reduction is accomplished by separating members of a population into homogeneous groups before sampling. Sampling each stratum reduces sampling error and can produce weighted means that have less variability than the arithmetic mean of a simple sampling of the population. [35]

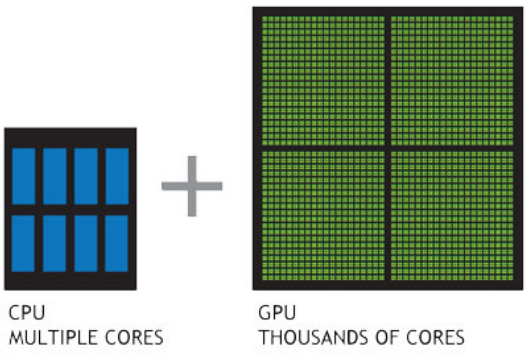
Modern research in the area of variance reduction techniques often includes a specific problem that requires a more focused study to utilize one of these previously described patterns. For example in the problem of atmospheric radiative transfer modeling Iwabuchi recently published work describing their proposal of some variance reduction techniques that they can use to help solve the problem of solar radiance calculations. They describe four methods that are developed directly from their problem. The first is to use a type of Russian roulette on values that will contribute small or meaningless amounts to the overall calculation within some threshold. Other methods include approximation methods for sharply peaked regions of the phase space, forcing collisions in under sampled regions, and numerical diffusion to smooth out noise. [36]

IV. State of the Art: GPU Research

In this section we will be looking at the recent advances in Monte Carlo research on GPU architectures. We will first look at different approaches people have taken to get onto the GPU. We will then look at Monte Carlo transport from the medical transport perspective in order compare approaches from the different communities. Then we will look at uses of ray tracing within a monte carlo transport application. Finally we will look at new algorithms choices through event based Monte Carlo transport.

A. *CPU Versus GPU*

"A simple way to understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously." [37]



[37]

A CPU has been developed from the beginning to optimize the performance of a single task. In order to accomplish this CPUs have been latency optimized, meaning that the time to complete one task, including gathering the necessary memory, has been reduced in any way possible. GPUs on the other-hand have been throughput optimized in order to complete as many tasks as possible in a given amount of time. This means that the time to complete a single task is most likely significantly longer than in a CPU but in a fixed amount of time the GPU will be able to accomplish many more tasks. So given a large enough number of tasks that can be accomplished in parallel the GPU can complete all of them faster.

B. *First Pass at GPU Computing*

This section will analyze the different approaches that people have taken to get their Monte Carlo transport codes working for GPU architectures. It will begin by comparing and contrasting different approaches by evaluating a few key areas of the studies that have been done: accuracy, performance and algorithmic choices. Following will be an evaluation of the effectiveness of the approaches for the range of problems being addressed. As a side note it is important to notice that I will be reporting speedups as reported by each paper on the hardware they were using at the time of their study.

Accuracy:

One of the first considerations the scientific community has when being introduced to a new computing platform is what levels of accuracy can they achieve with their simulation codes. Since the change from CPU to GPU computing brings a completely different hardware design it is important to understand how that design might affect the accuracy of any caluclations it is preforming. This concern was especially important in the early days of GPU computing when double precision was not supported and often even single precision answers would provide slightly different results. There are three key areas of accuracy that we will look at with these studies, being: Floating point precision, differences between CPU and GPU results, and IEE-754 compliance.

It was common that in early GPU computing when double floating point precision was not supported or supported well that people started thinking about GPU computing as not being accurate enough for their needs. Many early attempts at GPU computing includes discussions of accuracy in order

to validate the correctness of their results. While modern GPGPUs support double precision much better than before making much of the worry irrelevant, it is still important to consider the accuracy of a method that runs on a new hardware and may use a new algorithm.

Floating Point Accuracy : One of the primary concerns of the early GPU studies involved understanding the limits of floating point arithmetic on the GPU architecture. Nelson in his thesis work [38] describes one of his primary accuracy considerations as being the difference between single and double precision calculations. In older GPU hardware there was no support for double precision in the hardware and so in order to achieve double precision significantly more calculations were needed. In modern GPU hardware 64 bit double precision is becoming increasingly better supported and in the GPGPU cards there are dedicated double precision units and all of the necessary hardware changes required to include them.

Differences Between CPU and GPU results : More then the differences between single and double precision are also a concern for differences between the results that arise when using the same precision. This concern can be explained by understanding how floating-point math is accomplished on a computer [39]. There are two main reasons that differences arise. The first is that floating point mathematical operations that are done in a different order might produce a different result and sue to the nature of parallel computing often you cannot know or guarantee the order a set of calculations will be preformed in. The second reason is that modern day CPUs using x86 processors preform math internally on 80bit registers while a GPU does it on 32 bit (single precision) or 64 bit (double precision) registers. Because of this each math operation on a CPU might stay in registers and only be rounded down to 64 bits when it is saved to memory.

Jia et. al [40] showed that in their development of a Mont Carlo dose calculation code they could achieve speedups of 5 to 6.6 times their CPU version while maintaining within 1% of the dosing for more then 98% of the calculation points. They considered this adequate accuracy to consider using GPUs for doing these computations. Yepes et. al [41] also considered accuracy in their assessment of their GPU implementation. They concluded that in term of accuracy there was a good agreement between the dose distributions calculated with each version they ran, the largest discrepencies being only $\sim$3%, and so they could run the GPU version as accurately as any general-purpose Monte Carlo program. As these two groups have shown this amount of error is often very small and over the entire course of the simulation only brings 1-3% errors.

IEE-754 Compliance : Nelson discussed accuracy in his thesis work [38], stating that during the time of his work the floating-point arithmetic accuracy was not fully IEEE-754 compliant which opens the question of accuracy without a more fully featured test. Additionally, since NVIDIA as complete control over the implementation of floating point calculations on their GPUs their may be differences between generations that mitigate the usefulness of an accuracy study on one generation of hardware. Current generations of the NVIDIA GPU hardware are IEE-754 compliant however. In order to address issues of floating point accuracy they have even included a detailed description of the standard and the way CUDA follows the standard showing that at least while floating point accuracy is still a concern it is no more a concern than it was on a CPU implementation. [42]

Performance:

A second factor that is important to people making their first pass at GPU Monte Carlo is performance. Most early GPU studies emphasize the speedups between CPU and GPU as the primary advantage for moving over to the GPU hardware. Given the change in supercomputing designs these comparisons have become increasingly more important.

Often, performance is compared to the hardware maximmums such as peak of FLOPS or Memory Bandwidth. It is often assumed that an increase in available FLOPS will translate directly into incredible performance gains. In Lee et al.'s Debunking the 100X GPU vs. CPU myth [43], this discussion of performance is brought into new light showing the relative performance gains for different types of applications. The important thing to consider is the limiting factor between the hardware and the code. Because of this comparing current performance with that of peak performance is often very misleading.

The following discussions show the relative performances of Monte Carlo transport applications that underwent their initial transformations or studies to use the GPU hardware. We will not see he 100x performance that is often sought after, but instead we can understand the impact that each applications problem, algorithms, and implementation differences had on the performance as a whole.

Photon Transport: Badal and Badano [44] present work on photon transport in a voxelized geometry showing results around 27X over a single core CPU. Their work emphasizes simply using GPUs instead of CPUs and the advanatage as GPUs continue to increase in performance faster than CPUs.

Neutron Transport: Nelsons work presented in his thesis [38] shows a variety of models and considerations for his performance results. His work solving neutron transport considered multiple models for running the problem and optimizing for the GPU. The model that produced his best results shows 19.37X from a 49,152 neutrons per batch run for single precision. The same model shows 23.91X when using single precision and fast math, For double precision performance the model labeled model four had the fastest speedups with 11.21X and 12.66X with fast math.

Gamma Ray Transport: Work presented by Tickner [45] on X-ray and gamma ray transport uses a slightly modified scheme from the others by launching particles on a per block basis. In this way he hoped to remove the instruction level dependancies between particles running on the GPU hardware. In this work he showed he was capable of producing speedups of up to 35X over a single core cpu, and a significant improvement on any per-thread methods we have seen so far.

Coupled Electron Photon Transport: Jia et. al's work [40] in a dose calculation code for coupled electron photon transport follows a relatively straight forward algorithm. In their work they offload the data and computations to the GPU, simulate the particles, and then copy memory back. This method produced a modest performance increase on a GPU of around 5 to 6.6X over their runs on a CPU. The limitation of this speedup was attributed to the branching of the code and that effect it had on the GPU hardware.

Track Repeating Alogorithm: In contrast to Jia et al's work Yepes et al [41] showed that a different algorithm could greatly improve results. By converting a track-repeating algorithm instead of a full Monte Carlo, Yepes et al. gained aroung 75X the performance on the GPU over the CPU. It is thought that the simpler logic of this algorithm generated threads which followed closer logic to that of the algorithm presented in Jia at al's work.

Performance Evaluation: Throughout all of these examples one common theme can be seen. Performance can be gained doing Monte Carlo on the GPU. Performance can be more difficult to get due to the highly divergent nature of the full Monte Carlo application. Methods to deal with this divergence can show promising results that are worthy of further study. These outcomes are expected outcomes since Monte Carlo applications are embarrassingly parallel (good for GPUs) but also incredibly divergent (bad for GPUs).

In this section we see a wide range in performances, from as low as 5x to as high as 75x. While simplifications played a large role in the 75x algorithm we do see a full monte carlo application achieving speeds of 35x in the case of the work by Tickner [45]. I think that is important to note that while some of the differences in performance are due to the nature of each problem being solved, the algorithmic choices made can have a significant impact on the GPU implementations.

Algorithms:

Based on the performance studies we have just seen, it is important to highlight the algorithmic approaches that were taken so that we can understand the performances of each approach. If we can clearly find algorithms that show positive performance results than other codes can implement them for potential gain. In this section we are going to look closely at a few of the important or interesting algorithms we have seen attempted.

Monte Carlo transport applications tend to follow a simple model where each tracked particle is given its own thread and computations progress in an embarrassingly parallel fashion. On a GPU this also makes sense as a starting point since particles are independent and this progression leads to a simple natural parallel approach. It is often pointed out however that due to the divergent nature of Monte Carlo this approach might not be the best way organize Monte Carlo codes on GPU hardware.

Particle-Per-Block: We will first look at an alternative approach, the particle-per-block tracking algorithm described by Tickner [45]. First each tracked particle or quantum of radiation is given to a block of threads. Then calculations are performed for one particle on each block of threads. For example the particle intersection tests with the background geometry can be preformed in parallel on those threads for each piece of geometry that particle might be able to collide with. Areas where these parallel instructions can be utilized within a particles calculation are then used by the threads in a block computing for that particle.

This particle-per-block technique has shows promise as an effective way to counteract the divergence issue. Particles often diverge quite quickly from one another in the code paths they follow. This means that threads in a block are not always able to travel in lock step and can cause some serialization of the parallel regions. By using only one particle per block the divergence problem is nearly entirely removed from the equation. Additionally this method introduces new areas of parallelism that are not otherwise being taken advantage of, instruction level parallelism in the calculations for a single particle.

This method however, does not take full advantage of the parallelism in the hardware like those methods that do not mind the divergence do. Many threads can execute simultaneously at once within a block and only groupings of 32 threads are held in a WARP forced into the lockstep pattern that causes potential slowdowns. By running only one particle per block you are sacrificing some parallelism as not all tasks to calculate a particles path are parallel operations. Additionally, since warps are scheduled out of thread blocks any particle operations that are not done in parallel among the threads of a block are serializing themselves in a similar manner as to those algorithms that run one thread per particle waiting while divergent particles have a turn.

In summary I think that this method has some merit if it can find enough parallel work in the thread block to execute additional parallel tasks that would otherwise be stalled if following a simpler method. I also think that this method might end up showing the same characteristics of the simpler particle-per-thread model if the extra parallelism is not found, and instead loose out on the parallelism provided by particles that are not divergent from one another.

Event-Based Approaches: A second possibly more obvious method to escape the divergence issue is to switch particle tracking algorithms more dramatically from a history based version to an event based version. We will have a discussion of this further in the Section on event based algorithms later in this paper. Event based approaches require much more work then simply transforming an existing code to use the history based version on the GPU. And as Du et al discovered in their attempt at a event based Monte Carlo version of the Archer code [46] [47] [48] [49], getting any speedups with that method has a whole new host of challenges to overcome.

Voxelization Approaches: This method was used as for comparison on the GPUs. Voxelization of a geometry was done for each voxel and this process involved: ray-stabbing number counted on the GPU and then a parity-counting method was run on the CPU to detect if the voxel was inside the mesh

TABLE II
GPU SPEEDUP EVALUATION RESULTS

Case	Execution Time T_{CPU} (minutes)	Execution Time T_{GPU} (minutes)	Speed-up factor T_{CPU}/T_{GPU}
Neutron Transport Problem	0.496	0.017	29.2
eigen-value/criticality problem	4.25	0.5	8.5
Voxelization	2380.4	52.3	45.5

surface [50]. This method contained no divergence since all threads follow the exact same code paths. This process is often done to voxelize geometries for before Monte Carlo codes can be run. Doing this algorithm with no divergence produces a 45.5x speedup on the GPU over the CPU. This example was shows in Ding et al.'s evaluation report [51] in order to show the performance of the same GPU on different aspects related to Monte Carlo transport.

Evaluation:

A number of studies were conducted by groups identifying the potential benefits of GPU hardware but also the hardware and software issues when developing Monte Carlo applications. Among these concerns are memory limitations, lack of ECC support, lack of software optimization, limitations of SIMD architecture, clock speeds, and complex memory allocation schemes. In addition the achieved performance was often not more than could be gotten with unchanged codes on a cluster. In some cases though speedups were large and easy to achieve such as the 45X speedup of the voxelized approach. The results from Ding et al.'s evaluations can be seen in Table II. The only strong conclusion from these works are that a clear and defined path are not yet known on how to take full advantage of the available parallelism without suffering performance penalties in turn. [51]

C. Monte Carlo and Medicine

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

D. Monte Carlo and Ray Tracking

One important and often computationally expensive aspect of Monte Carlo transport is the step that determines if the particle will collide with any background geometry, or at least cross into a different material zone. This is done in a very similar way to the visualization technique known as ray tracing. Ray racing is a technique in computer graphics for "generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects" [52].

The general process of ray tracing is very similar to Monte Carlo transport in the need to do many intersection tests and from potentially scattered sources. Bergmann decided to study the potential of using the power of a highly optimized GPU library, OptiX [**?**]. OptiX is a scalable framework for building ray tracing applications [53] [54].

The first study conducted was to determine the optimum configuration for OptiX as well as the capability for OptiX to be initialized with random starting points and directions as is most likely to be the case in a Monte Carlo application. When using a ray tracing library it is important to consider the two areas that can scale: the number of concurrently traced rays and the number of geometrical objects in the scene. Since nuclear reactor simulations might contain thousands of material zones in complex geometric layouts; knowing this last scaling parameter is especially important to not overlook. [**?**]. In these studies after reaching 10^6 particles the rates became fairly consistent. Bergmann also notes some important points, such as which acceleration structure was always best and when memory become a constraint on the problem that could be run. The conclusion from this study was that OptiX could be used to handle the geometry representation in a Monte Carlo neutron transport code. Additionally, for best performance one should use a primitive-based geometry instancing method, a BVH acceleration structure, and run as many parallel rays as possible.

In addition to the use of a pre-existing tool like NVIDIA's OptiX library, other groups looked at optimizing Mont Carlo transport by focusing on treating it like a ray tracing problem. Xiao et al. [55] focused on the data locality issues in all ray tracing applications on GPUs. They describe a new data locality method based on task partitioning and scheduling in order to enhance spacial and temporal data locality by ordering random rays into coherent groups. By applying this method they achieved a 6-8X speedup over the previous GPU version of radiation therapy Monte Carlo transport.

These examples show that progress in connected fields can positively impact the applications in Monte Carlo transport. Ray tracing is only one aspect of a full Monte Carlo transport application but as we have seen here it can be greatly beneficial to look at work done in these related fields and bring those ideas back into the full application.

E. Event Based Techniques

Much discussion has been aimed at the negative affect divergence in Monte Carlo codes has on performance. Given the inherently parallel nature of the algorithm, each particle being tracked independently, performance of Monte Carlo transport codes on the GPU should be incredible. We often see the opposite and have seen throughout this section the different approaches and often marginal speedups that were attained. One main concern with many of these studies was the affect divergence had on their algorithm.

In order to combat divergence, a old scheme was re-evaluated for use on GPU architectures. Given some of the similarities between the classic vector machines of the 1980's/90's with modern GPU hardware, it is reasonable to consider some of those algorithms for use now. One main approach that worked well on SIMD vector hardware, is the event-based approach. In the event-based approach particles are processed in groups that are performing the same event. There are multiple variations to this idea and a few of those are presented here.

Vectorized Algorithm

Early event based algorithms were designed for vector machines and were called vectorized algorithms. Martin describes a successful vectorized algorithm as well some variations in his paper [56]. The conventional Monte Carlo algorithm cannot be vectorized since treating many histories simultaneously would immediately fail after the first step of the simulation as each particle can undergo a different event. In order to achieve vectorization the histories need to be split into events, which are similar and can be processed in a vectorized manner, i.e. the same set of instructions. The basic event based iteration algorithm is descirbed in Algorithm 3.

In addition to the basic event based approach there are a few variations discussed in Martins paper that expand on this model. One variation is the stack-driven approach. In this approach the events are further divided into smaller computational tasks. Instead of cycling through the tasks in a fixed order, the computation can move forward by selecting the event with the largest number of particles. This involves a tradeoff of simplified control flow for maximizing the vector lengths of the computational components.

Vectorized versions of the Monte Carlo transport algorithms are all based on this original based algorithm. There are many variations but the principal differences all depend on the methods used for organizing and treating the vectors of particles. There are variations using stack, tags, and tasks. The major downside to the event based approach is that is requires a large change to the source code.

Event Based for GPU

Event based methods used for the GPU follow similar design patterns as those that were developed for vector machines. One prime example is the event based version developed by Bergmann for the code WARP [57]. Figure IV-E outlines the inner transport loop broken into its separate stages. Figure IV-E outlines the outer transport loop between neutron batches.

Bergmann's event based Monte Carlo code WARP [57] utilizes a series of kernels that each solves one piece of the process. Once the each neutron knows which path it will go down – i.e. scattering, fission, etc. – each of those possible paths is launched in a seperate kernel. Unlike the basic vectorized approach or the stack based approach however, all of the events are launched at once using concurrent kernels due to CUDA streaming properties. In this way the main divergent

Algorithm 3: The basic iteration event

for *event n = 0, 1, 2, ...* **do**
- · Fetch Γ^n
- · Preform free flight analysis:
  - · gather the cross section data and geometry data tabulated by particle,
  - · $\Sigma \leftarrow$ S,
  - · $\rho \leftarrow$ R;
  - · using Σ, sample a vector of distances to collision, d_c
  - · using ρ, determine vector of minimum distances to boundary, d_b
  - · determine the minimum distances to the end of event,
    - d_{min} = min$[d_c, d_b]$;
  - · update the particle coordinates,
    - $r^{n+1} = r^n + \Omega^n \cdot r_{min}$
- · Perform collision analysis:
  - · gather particle attributes,
    - $\Omega \leftarrow \Gamma^n, E \leftarrow \Gamma^n$;
  - · evaluate collision physics for new direction cosines and energies,
    - $\Omega' \leftarrow \Omega$, $E' \leftarrow E$
  - · scatter new particle attributes back into bank,
    - $\Omega' \leftarrow \Gamma^n$, $E' \leftarrow \Gamma^n$
- · Perform the boundary analysis:
  - · gather particle zone indices Z,
    - $Z \leftarrow \Gamma^n$
  - · determine new zone indices,
    - $Z' \leftarrow Z$:
  - · scatter new zone indices back into bank.
    - $Z' \rightarrow \Gamma^n$
- · Update the particle bank,
  - $\Gamma^n \Rightarrow \Gamma^{n+1}$ $(with\ L_{n+1}\ particles)$
  - (e.g. compress out terminated particles).
- · If $L_{n+1} \neq 0, continue$

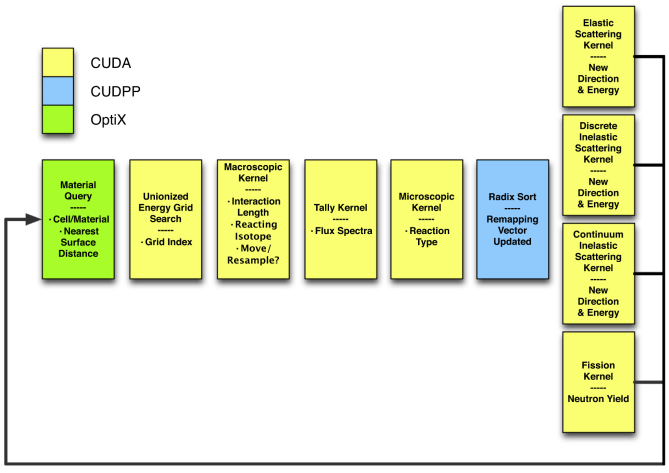


Fig. 1. WAP inner tranport loop that is executed until all neutrons in a batch are completed [57]

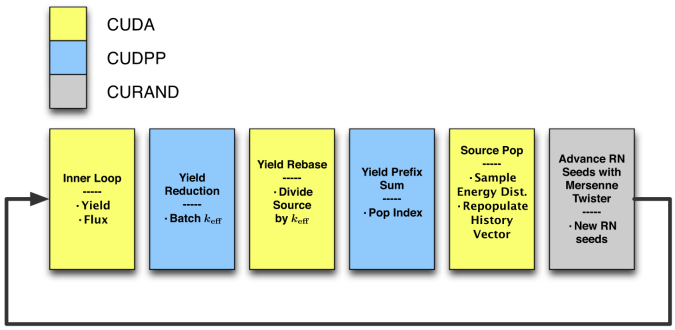


Fig. 2. WAP inner tranport loop that is executed until all neutrons in a batch are completed [57]

part of the code is broken into relatively non-divergent kernels which are then launched simultaneously so as to continue to utilize the full hardware.

Not all attempts at vectorization, or implementing an event based algorithm for Monte Carlo transport codes has been successful however. Liu [58] describes an event based approach that after being implemented produced a roughly ten times slower version then the history based code. This example shows how complicated the task of implementing an event-based algorithm can be, and that it is possible as well that not al Monte Carlo transport problems can be solved efficiently in an event-based fashion. Liu attributed their slow down to the memory access latency due to the high amount of global memory transactions and showed that the cost of this in an event based method did not outweigh the benefit of reducing thread divergence and increasing warp execution efficiency.

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 [59]. 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 co-processors (e.g., Intel Xeon Phi in Tianhe-2), and large multi-core CPUs (e.g., IBM Power, Intel Xeon in Tianhe-2 and others) [59], [60], [61]. 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) [61]. Given this wide array of possible architectures the value of portable performance has never before been so high.

A. *Abstraction Layers*

OpenMP

[62] [63] [64]

OpenACC

[65] [66] [67]

Thrust

[68] [69]

RAJA

[70] [71]

Kokkos

[72] [73]

Chapel

[74] [75]

VTK-m

[76] [77]

Dax: [78] [79]

VI. Monte Carlo and Portable Performance

In this section Monte Carlo transport applications will be evaluated as a platform for portable performance studies. Monte Carlo transport offers a unique set of challenges and interesting lessons to the world of portable performance possibilities. Already discussed are efforts in Ray Tracing done by using the EAVL/VTK-m framework, and the successes seen there. In addition, the idea of portable performance is a popular one with many different groups putting forward possible designs and library options. Each of these abstraction layers– EAVL, DAX, PISTON, VTK-m, Thrust, OpenMP, OpenACC, RAJA, Chapel, and Kokkos –has its own plusses and minuses, as well as different levels of effort required to make it function in already existing codes.

There is a large number of potential directions and possible areas to focus on in this study. Looking now at Bleile's work with a Monte Carlo mini app, ALPSMC [] [], the Mont Carlo algorithmic question of event based versus history based algorithms in a simplified Monte Carlo code, while also studying the effects different parallel paradigms have on performance and ease of code conversion.

A. *ALPSMC*

ALPSMC is a Monte Carlo test code that models neutron transport in a one dimensional planar geometry, through a binary stochastic medium. It is originally a serial C++ application that follows an all particle history based approach. This history based algorithm is shown in Algorithm 4

The original work was to convert this algorithm into an event based approach. The event based algorithm performs data parallel operations across all of the particles that are in the same event, as well as a series of data parallel steps required to do proper book keeping to get the particles for each event. The event based algorithm is defined in Algorithm 5. The operations required to launch the event kernels are defined as follows and correspond to lines 6 and 7 of Algorithm 5. "

Step 1: thrust::transform — Fill out a stencil map of 1's and 0's of all particles doing event E (where each particle whose next event is E will get a 1 in the stencil map at its index location)

Algorithm 4: History-based Monte Carlo algorithm

```
1  foreach particle history do
2      generate particle from boundary condition or source
3      while particle not escaped or absorbed do
4          sample distance to collision in material
5          sample distance to material interface
6          compute distance to cell boundary
7          select minimum distance, move particle, and
           perform event
8          if particle escaped spatial domain then
9              update leakage tally
10             end particle history
11         if particle absorbed then
12             update absorption tally
13             end particle history
```

Algorithm 5: Event-based Monte Carlo algorithm

```
1  foreach batch of particle histories (fits in memory
   constraint) do
2      generate all particles in batch from boundary
       condition or source
3      determine next event for all particles (collision,
       material interface crossing, cell boundary crossing)
4      while particles remaining in batch do
5          foreach event E in (collision, material interface
           crossing, cell boundary crossing) do
6              identify all particles whose next event is E
7              perform event E for identified particles and
               determine next event for these particles
8          if particle escaped spatial domain then
9              update leakage tally
10         if particle absorbed then
11             update absorption tally
12         delete particles absorbed or leaked
```

Step 2: thrust::reduce — Count the number of elements labeled 1 in the stencil (determines the number of particles that will perform event E)

Step 3: Check if the number of elements is greater than 0 (check if any particles are performing event E)

Step 4: thrust::exclusive_scan — generate indices for index mapping from stencil map (indices for each particle performing event E)

Step 5: Allocate a new map of appropriate size (map to hold indices for all particles performing event E)

Step 6: Scatter indexes from scan into new index map (reduces the exclusive_scan generated indices into the map that holds only enough for particles performing event E)

Step 7: Use new index map in permutation_iterator loops over all particles (combining the index map with the permutation iterator allows loops over all particles to operate only on the particles selected in the index map)

" []

Thrust was chosen as the platform for implementing the data parallel operations, though many of the options discussed would have worked for this. The key design choice was that each operation can be done using data parallel primitives. This study also use a direct CUDA implementation that launched kernels for the main events and used thrust for data management.

The performance results from the initial implementation [] were varied, with the best CUDA version reaching about 12x and the thrust OpenMP version reaching 2.2x with 16 threads. The optimized implementation discovered a few areas for improvement and achieved fairly significant results. Table III shows the summary of ALPSMC speedup results.

Through this experience we have demonstrated a few important points. First we demonstrated excellent performance in both algorithms, achieving $\tilde{5}$0x performance for both history and event based algorithms. Additionally, this examples showed that using data parallel design and an abstraction layer – in this case Thrust – can preform just as well as when programmed in native CUDA under these circumstances. Lastly, while the GPU performance is high the CPU performance is still lacking. Figure VI-A shows that the OpenMP version scales well but it has a much higher overhead than the original serial version.

TABLE III
MAXIMUM SPEEDUPS FOR EACH APPROACH WHEN COMPARED TO THE ORIGINAL HISTORY-BASED SERIAL METHOD

Method	Speedup
CUDA Event SOA	31.32
CUDA History 52.78	
Thrust Event CUDA SOA	54.62
Thrust Event OpenMP SOA	5.54

All of these considerations lay a foundation for future study. ALPSMC is a small test code that functions like a mini app to

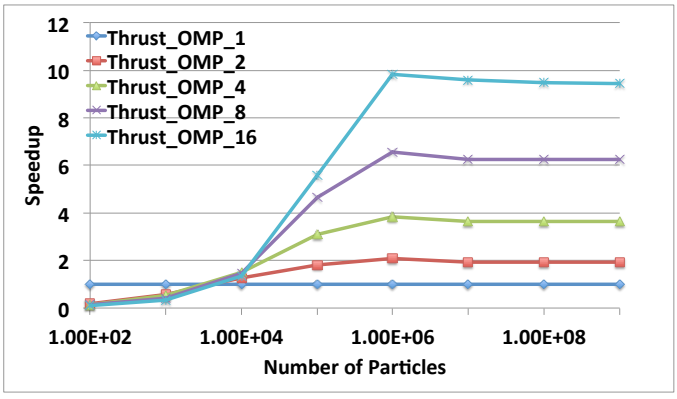


Fig. 3. Speedups versus number of particles for the event based Thrust CPU method with 1, 2, 4, 8, 16 OpenMP threads compared to the Thrust CPU method serially. []

a larger application. Since ALPSMC does simplify many of the complex parts of the Monte Carlo transport problem the actual speedups may not accurately reflect the final performance possibilities of a more fully featured Monte Carlo transport code.

B. Proposal

This survey has outlined the long history of Monte Carlo transport applications and research. In this survey we have seen that there has been a major revival of computational study with every new generation of supercomputing platform. Monte Carlo transport applications have only one significant gap in knowledge were groups have just recently started adding supporting research. That area is the case of a fully features Monte Carlo application on GPUs and scalable to high numbers of GPUs as we will see on the next generation of supercomputers.

Previous thesis work lays a starting foundation to build upon, but does not begin to provide a full solution for other groups to follow. One significant hurdle in the Monte Carlo transport world is the necessary redesign for GPU hardware. It is important for groups to not need to rewrite their entire applications, which can consist of hundreds of thousands of lines of code. Codes like MCNP [80] [81], Mercury [82], and even OpenMC [83] will all be faced with many decisions on how to progress into the future of computing.

The OpenMC group has started looking into this problem through the use of mini applications, RSBench [84] and XSBench [85]. This approach allows groups to focus on the specific areas that are important to them and optimize an easier to manipulate application before attempting any changes on a full scale production application. One of the major focuses of this work has been in the area of continuous energy cross section searches, since for a large number of their problems that functionality took $\tilde{8}5\%$ of their workload.

As with many other groups the Mercury group at LLNL has also begun work on a mini application. This application is different to RSBench and XSBench in multiple ways but most significantly it uses the multi group energy cross sections and emphasizes many of the key areas of the Mercy production application. This application will provide a begining point for redesign of the Mercury Monte Carlo code and allow for a rich research environment for the upcomming years.

I propose to work under the Mercury group on the Quicksilver mini application, in order to develop a scalable GPU version of the code in a way that can translate to direct modifications of the production application. I will further explore the event versus history based dilemma under the Quicksilver application in order to understand the potential performance when compared to the much larger necissary redesign. I also plan on exploring the potential for a hybrid event/history algorithm that might utilize the advantages of both when possible. I also plan on using the data parallel primitive design scheme as well as some layer of abstraction to make my research portable to multiple architecture platforms. Additionally, there are a number of optimizations, data structures, and smaller research problems to tackle along the path of development. Finally, I plan on summarizing the results of my research through large scale testing on the Trinity MIC platform, as well as the not yet released Sierra NVIDIA Volta platform.

The goal of this research will be to provide a concrete path forward for the Mercury team as well as provide a mini application that scales well on both of the competing top architectures at once. This path involved many as of yet unanswered questions and a clear path of research for providing new an unique research to this field.

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] F. B. Brown, "Recent advances and future prospects for monte carlo," *Progress in nuclear science and technology*, vol. 2, pp. 1–4, 2011.

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] Y. Wang, E. Brun, F. Malvagi, and C. Calvin, "Competing energy lookup algorithms in monte carlo neutron transport calculations and their optimization on cpu and intel mic architectures," *Procedia Computer Science*, vol. 80, pp. 484–495, 2016.

[26] F. B. Brown, "New hash-based energy lookup algorithm for monte carlo codes," *Trans. Am. Nucl. Soc*, vol. 111, pp. 659–662, 2014.

[27] J. Leppänen, "Two practical methods for unionized energy grid construction in continuous-energy monte carlo neutron transport calculation," *Annals of Nuclear Energy*, vol. 36, no. 7, pp. 878–885, 2009.

[28] A. Lund and A. Siegel, "Using fractional cascading to accelerate cross section lookups in monte carlo neutron transport calculations," in *ANS M&C 2015*, LaGrange Park, IL, 2015.

[29] H. Kahn and A. W. Marshall, "Methods of reducing sample size in monte carlo computations," *Journal of the Operations Research Society of America*, vol. 1, no. 5, pp. 263–278, 1953.

[30] "Variance reduction." [Online]. Available: https://en.wikipedia.org/wiki/variance_reduction

[31] "Antithetic variates." [Online]. Available: https://en.wikipedia.org/wiki/Antithetic_variates

[32] "Control variates." [Online]. Available: https://en.wikipedia.org/wiki/Control_variates

[33] "Importance sampling." [Online]. Available: https://en.wikipedia.org/wiki/Importance_sampling

[34] P. Melnik-Melnikov and E. Dekhtyaruk, "Rare events probabilities estimation by ?russian roulette and splitting? simulation technique," *Probabilistic engineering mechanics*, vol. 15, no. 2, pp. 125–129, 2000.

[35] "Stratified sampling." [Online]. Available: https://en.wikipedia.org/wiki/Stratified_sampling

[36] H. Iwabuchi, "Efficient monte carlo methods for radiative transfer modeling," *Journal of the atmospheric sciences*, vol. 72, no. 9, 2015.

[37] "What is gpu computing?" [Online]. Available: http://www.nvidia.com/object/what-is-gpu-computing.html

[38] A. G. Nelson, "Monte carlo methods for neutron transport on graphics processing units using cuda," Ph.D. dissertation, The Pennsylvania State University, 2009.

[39] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[40] X. Jia, X. Gu, J. Sempau, D. Choi, A. Majumdar, and S. B. Jiang, "Development of a gpu-based monte carlo dose calculation code for coupled electron–photon transport," *Physics in medicine and biology*, vol. 55, no. 11, p. 3077, 2010.

[41] P. P. Yepes, D. Mirkovic, and P. J. Taddei, "A gpu implementation of a track-repeating algorithm for proton radiotherapy dose calculations," *Physics in medicine and biology*, vol. 55, no. 23, p. 7107, 2010.

[42] "Floating point and ieee 754," Sep 2015. [Online]. Available: http://docs.nvidia.com/cuda/floating-point/#axzz4k4zi4wrv

[43] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund *et al.*, "Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 451–460, 2010.

[44] A. Badal and A. Badano, "Accelerating monte carlo simulations of photon transport in a voxelized geometry using a massively parallel graphics processing unit," *Medical physics*, vol. 36, no. 11, pp. 4878–4880, 2009.

[45] J. Tickner, "Monte carlo simulation of x-ray and gamma-ray photon transport on a graphics-processing unit," *Computer Physics Communications*, vol. 181, no. 11, pp. 1821–1832, 2010.

[46] X. G. Xu, T. Liu, L. Su, X. Du, M. Riblett, W. Ji, D. Gu, C. D. Carothers, M. S. Shephard, F. B. Brown *et al.*, "Archer, a new monte carlo software tool for emerging heterogeneous computing environments," *Annals of Nuclear Energy*, vol. 82, pp. 2–9, 2015.

[47] X. Du, T. Liu, W. Ji, X. Xu, and F. Brown, "Evaluation of vectorized monte carlo algorithms on gpus for a neutron eigenvalue problem," American Nuclear Society, 555 North Kensington Avenue, La Grange Park, IL 60526 (United States), Tech. Rep., 2013.

[48] T. Liu, X. G. Xu, and C. D. Carothers, "Comparison of two accelerators for monte carlo radiation transport calculations, nvidia tesla m2090 gpu and intel xeon phi 5110p coprocessor: A case study for x-ray ct imaging dose calculation," *Annals of Nuclear Energy*, vol. 82, pp. 230–239, 2015.

[49] L. Su, X. Du, T. Liu, and X. Xu, "Monte carlo electron-photon transport using gpus as an accelerator: Results for a water-aluminum-water phantom," American Nuclear Society, 555 North Kensington Avenue, La Grange Park, IL 60526 (United States), Tech. Rep., 2013.

[50] Y. H. Na, B. Zhang, J. Zhang, P. F. Caracappa, and X. G. Xu, "Deformable adult human phantoms for radiation protection dosimetry: anthropometric data representing size distributions of adult worker populations and software algorithms," *Physics in medicine and biology*, vol. 55, no. 13, p. 3789, 2010.

[51] A. Ding, T. Liu, C. Liang, W. Ji, M. S. Shephard, X. G. Xu, and F. B. Brown, "Evaluation of speedup of monte carlo calculations of two simple reactor physics problems coded for the gpu/cuda environment," 2011.

[52] "Ray tracing (graphics)." [Online]. Available: https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

[53] "Optix programming guide." [Online]. Available: http://docs.nvidia.com/gameworks/content/gameworkslibrary/optix/optix_programming_guide.htm

[54] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison *et al.*, "Optix: a general purpose ray tracing engine," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4. ACM, 2010, p. 66.

[55] K. Xiao, D. Z. Chen, X. S. Hu, and B. Zhou, "Monte carlo based ray tracing in cpu-gpu heterogeneous systems and applications in radiation therapy," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 247–258.

[56] W. R. Martin, "Successful vectorization-reactor physics monte carlo code," *Computer Physics Communications*, vol. 57, no. 1-3, pp. 68–77, 1989.

[57] R. M. Bergmann, "The development of warp-a framework for continuous energy monte carlo neutron transport in general 3d geometries on gpus," 2014.

[58] T. Liu, X. Du, W. Ji, X. G. Xu, and F. B. Brown, "A comparative study of history-based versus vectorized monte carlo methods in the gpu/cuda environment for a simple neutron eigenvalue problem," in *SNA+ MC 2013-Joint International Conference on Supercomputing in Nuclear Applications+ Monte Carlo*. EDP Sciences, 2014, p. 04206.

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

[60] "June 2016," Top 500 The List, jun 2016, https://www.top500.org/lists/2016/06/.

[61] H. Childs, "Portable performance," 2015, http://cdux.cs.uoregon.edu/portable_perf.html.

[62] "Openmp." [Online]. Available: http://www.openmp.org/

[63] S. Lee, S.-J. Min, and R. Eigenmann, "Openmp to gpgpu: a compiler framework for automatic translation and optimization," *ACM Sigplan Notices*, vol. 44, no. 4, pp. 101–110, 2009.

[64] E. Ayguadé, R. M. Badia, P. Bellens, D. Cabrera, A. Duran, R. Ferrer, M. González, F. Igual, D. Jiménez-González, J. Labarta *et al.*, "Extending openmp to survive the heterogeneous multi-core era," *International Journal of Parallel Programming*, vol. 38, no. 5-6, pp. 440–459, 2010.

[65] S. Wienke, P. Springer, C. Terboven, and D. an Mey, "Openacc?first experiences with real-world applications," in *European Conference on Parallel Processing*. Springer, 2012, pp. 859–870.

[66] "Openacc." [Online]. Available: http://www.openacc.org/

[67] Y. Wang, Q. Qin, S. C. W. SEE, and J. Lin, "Performance portability evaluation for openacc on intel knights corner and nvidia kepler," *HPC China*, 2013.

[68] J. Hoberock and N. Bell, "Thrust," 2008.

[69] ——, "Thrust: A parallel template library," *Online at http://thrust. googlecode. com*, vol. 42, p. 43, 2010.

[70] R. Hornung, J. Keasler *et al.*, "The raja portability layer: overview and status," *Lawrence Livermore National Laboratory, Livermore, USA*, 2014.

[71] R. Hornung, J. Keasler, A. Kunen, H. Jones, and D. Beckingsale, "Raja-llnl hpc architecture portability encapsulation layer version 1.0," Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States), Tech. Rep., 2016.

[72] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202–3216, 2014.

[73] H. C. Edwards, D. Sunderland, V. Porter, C. Amsler, and S. Mish, "Manycore performance-portability: Kokkos multidimensional array library," *Scientific Programming*, vol. 20, no. 2, pp. 89–114, 2012.

[74] B. L. Chamberlain, D. Callahan, and H. P. Zima, "Parallel programmability and the chapel language," *International Journal of High Performance Computing Applications*, vol. 21, no. 3, pp. 291–312, 2007.

[75] A. Sidelnik, S. Maleki, B. L. Chamberlain, M. J. Garzar'n, and D. Padua, "Performance portability with the chapel language," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 582–594.

[76] K. D. Moreland, "Vtk-m overview (nvidia design review)." Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2015.

[77] K. D. Moreland, R. K. Maynard, C. L. Sewell, and J. O. Meredith, "Vtk-m: A foundation (sdav meeting 2014-07)." Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2014.

[78] K. Moreland, U. Ayachit, B. Geveci, and K.-L. Ma, "Dax: Data analysis at extreme."

[79] ——, "Dax toolkit: A proposed framework for data analysis and visualization at extreme scale," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*. IEEE, 2011, pp. 97–104.

[80] T. Goorley, M. James, T. Booth, F. Brown, J. Bull, L. Cox, J. Durkee, J. Elson, M. Fensin, R. Forster *et al.*, "Initial mcnp6 release overview," *Nuclear Technology*, vol. 180, no. 3, pp. 298–315, 2012.

[81] E. Padovani, S. Pozzi, S. Clarke, and E. Miller, "Mcnpx-polimi user?s manual," *C00791 MNYCP, Radiation Safety Information Computational Center, Oak Ridge National Laboratory*, vol. 1, 2012.

[82] P. Brantley, S. Dawson, M. McKinley, M. O'Brien, D. Stevens, B. Beck, E. Jurgenson, C. Ebbers, and J. Hall, "Recent advances in the mercury monte carlo particle transport code," in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013)*, 2013, pp. 5–9.

[83] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, "Openmc: A state-of-the-art monte carlo code for research and development," *Annals of Nuclear Energy*, vol. 82, pp. 90–97, 2015.

[84] J. R. Tramm, A. R. Siegel, B. Forget, and C. Josey, "Performance analysis of a reduced data movement algorithm for neutron cross section data in monte carlo simulations," in *International Conference on Exascale Applications and Software*. Springer, 2014, pp. 39–56.

[85] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis," *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.

[86] P. K. Romano and B. Forget, "The openmc monte carlo particle transport code," *Annals of Nuclear Energy*, vol. 51, pp. 274–281, 2013.

[87] M. McKinley and B. Beck, "Implementation of the generalized interaction data interface (gidi) in the mercury monte carlo code," in *Proceedings of ANS MC2015-Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Nashville, Tennessee*, 2015.

[88] J. L. Vujic and W. R. Martin, "Vectorization and parallelization of a production reactor assembly code," *Progress in Nuclear Energy*, vol. 26, no. 3, pp. 147–162, 1991.

[89] M. Larsen, J. S. Meredith, P. A. Navrátil, and H. Childs, "Ray tracing within a data parallel framework," in *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2015, pp. 279–286.

[90] M. Larsen, S. Labasan, P. A. Navrátil, J. S. Meredith, and H. Childs, "Volume rendering via data-parallel primitives." in *EGPGV*, 2015, pp. 53–62.

[91] K. Moreland, M. Larsen, and H. Childs, "Visualization for exascale: Portable performance is critical," *Supercomputing frontiers and innovations*, vol. 2, no. 3, pp. 67–75, 2015.

[92] H. C. Edwards and D. Sunderland, "Kokkos array performance-portable manycore programming model," in *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*. ACM, 2012, pp. 1–10.

[93] L.-t. Lo, C. Sewell, and J. P. Ahrens, "Piston: A portable cross-platform framework for data-parallel visualization operators." in *EGPGV*, 2012, pp. 11–20.

[94] S. Lee and R. Eigenmann, "Openmpc: Extended openmp programming and tuning for gpus," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–11.

[95] P. Després, J. Rinkel, B. H. Hasegawa, and S. Prevrhal, "Stream processors: a new platform for monte carlo calculations," in *Journal of Physics: Conference Series*, vol. 102, no. 1. IOP Publishing, 2008, p. 012007.

[96] C. Gong, J. Liu, B. Yang, L. Deng, G. Li, X. Li, Q. Hu, and Z. Gong, "Accelerating mcnp-based monte carlo simulations for neutron transport on gpu," *International Journal of Radiation Oncology* Biology* Physics*, vol. 81, no. 2, pp. S157–S158, 2011.

[97] N. Ren, J. Liang, X. Qu, J. Li, B. Lu, and J. Tian, "Gpu-based monte carlo simulation for light propagation in complex heterogeneous tissues," *Optics express*, vol. 18, no. 7, pp. 6811–6823, 2010.

[98] M. Daga, Z. S. Tschirhart, and C. Freitag, "Exploring parallel programming models for heterogeneous computing systems," in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 98–107.

[99] G. E. Blelloch, *Vector models for data-parallel computing*. MIT press Cambridge, 1990, vol. 356.

[100] R. Rahaman, D. Medina, A. Lund, J. Tramm, T. Warburton, and A. Siegel, "Portability and performance of nuclear reactor simulations on many-core architectures," in *Proceedings of the 3rd International Conference on Exascale Applications and Software*. University of Edinburgh, 2015, pp. 42–47.

[101] E. Alerstam, T. Svensson, and S. Andersson-Engels, "Parallel computing with graphics processing units for high-speed monte carlo simulation of photon migration," *Journal of biomedical optics*, vol. 13, no. 6, pp. 060 504–060 504, 2008.

[102] Q. Xu, G. Yu, and K. Wang, "Research on gpu acceleration for monte carlo criticality calculation," in *SNA+ MC 2013-Joint International Conference on Supercomputing in Nuclear Applications+ Monte Carlo*. EDP Sciences, 2014, p. 04210.

[103] Q. Xu, G. Yu, X. Wu, and K. Wang, "A gpu-based local acceleration strategy for monte carlo neutron transport," *Transactions of the American Nuclear Society*, vol. 107, p. 526, 2012.

[104] G. Bosilca, A. Bouteiller, T. Herault, P. Lemarinier, N. O. Saengpatsa, S. Tomov, and J. J. Dongarra, "Performance portability of a gpu enabled factorization with the dague framework," in *2011 IEEE International Conference on Cluster Computing*. IEEE, 2011, pp. 395–402.

[105] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, "From cuda to opencl: Towards a performance-portable solution for multi-platform gpu programming," *Parallel Computing*, vol. 38, no. 8, pp. 391–407, 2012.

[106] P. J. Crossno, J. Patchett, E. Brugger, C. Guilbaud, and T. Carrard, "Nnsa/cea cooperation in computer science: Visualization." Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2015.

[107] T. Scudiero, "Monte carlo neutron transport - simulating nuclear reactions one neutron at a time," in: GPU Technology Conference. San Jose, California, 2014.