Parallel Fission Bank Algorithms in Monte Carlo Criticality Calculations

Parallel Fission Bank Algorithms in Monte Carlo Criticality Calculations

Paul K. Romano*
Massachusetts Institute of Technology
Department of Nuclear Science and Engineering
77 Massachusetts Avenue, Building 24-607
Cambridge, MA 02139

Benoit Forget
Massachusetts Institute of Technology
Department of Nuclear Science and Engineering
77 Massachusetts Avenue, Building 24-214
Cambridge, MA 02139

Abstract

In this work, we describe a new method for parallelizing the source iterations in a Monte Carlo criticality calculation. Instead of having one global fission bank that needs to be synchronized as is traditionally done, our method has each processor keep track of a local fission bank while still preserving reproducibility. In doing so, it is required to send only a limited set of fission bank sites between processors, thereby drastically reducing the total amount of data sent through the network. The algorithm was implemented in a simple Monte Carlo code and shown to scale up to hundreds of processors and furthermore outperforms traditional algorithms by at least two orders of magnitude in wall-clock time.

Keywords: Monte Carlo, eigenvalue calculation, parallel

*Email: romano7@mit.edu

1 Introduction

The ability to simulate complex transport phenomena using stochastic methods was recognized early on in the development of multiplying fission systems. Also recognized was the fact that while providing an elegant means of computing functionals, such methods would require a great amount of computation as well. The development of Monte Carlo methods has thus gone hand-in-hand with the development of computers over the course of the last half century.

Due to the computationally-intensive nature of Monte Carlo methods, there has been an ever-present interest in parallelizing such simulations. Even in the first paper on the Monte Carlo method [1], John Metropolis and Stanislaw Ulam recognized that solving the Boltzmann equation with the Monte Carlo method could be done in parallel very easily whereas the deterministic counterparts for solving the Boltzmann equation did not offer such a natural means of parallelism. With the introduction of vector computers in the early 1970s, general-purpose parallel computing became a reality. In 1972, Troubetzkoy *et al.* designed a Monte Carlo code to be run on the first vector computer, the ILLIAC-IV [2]. The general principles from that work were later refined and extended greatly through the work of Forrest Brown in the 1980s [3]. However, as Brown's work shows, the single-instruction multiple-data (SIMD) parallel model inherent to vector processing does not lend itself to the parallelism on particles in Monte Carlo simulations. Troubetzkoy *et al.* recognized this, remarking that "the order and the nature of these physical events have little, if any, correlation

from history to history," and thus following independent particle histories simultaneously using a SIMD model is difficult.

The difficulties with vector processing of Monte Carlo codes led to the adoption of the single-process multiple data (SPMD) technique for parallelization. In this model, each different process tracks a particle independently of other processes, and between source iterations the processes communicate data through a message-passing interface. This means of parallelism was enabled by the introduction of message-passing standards in the late 1980s and early 1990s such as PVM and MPI. The SPMD model proved much easier to use in practice and took advantage of the inherent parallelism on particles rather than instruction-level parallelism. As a result, it has since become ubiquitous for Monte Carlo simulations of transport phenomena.

Thanks to the particle-level parallelism using SPMD techniques, extremely high parallel efficiencies could be achieved in Monte Carlo codes. Until the last decade, even the most demanding problems did not require transmitting large amounts of data between processors, and thus the total amount of time spent on communication was not significant compared to the amount of time spent on computation. However, today's computing power has created a demand for increasingly large and complex problems, requiring a greater number of particles to obtain decent statistics (and convergence in the case of eigenvalue calculations). This results in a correspondingly higher amount of communication, potentially degrading the parallel efficiency.

As an example, we consider the simulation of a full-core PWR using Monte Carlo methods. Hoogenboom and Martin recently proposed a full-core benchmark problem [4] that will serve as a good example for our pur-

poses. This benchmark model has 241 assemblies, with each assembly containing a 17 by 17 square rod array. As a simplification, no control rods are modeled and thus all 289 pins in an assembly are fuel. In Kelly *et al.*'s analysis of the Hoogenboom-Martin benchmark problem using the MC21 Monte Carlo code, 100 evenly spaced axial nodes were used over the 400 cm length of each fuel pin [5]. Thus, one would need 6 964 900 different materials in order to deplete each fuel region individually. If we go one step further and subdivide each axial node radially to resolve the difference in flux due to spatial self-shielding, this would add proportionally many tally regions and materials. With three radial zones, we would then need 20 894 700 materials. To obtain good statistics would require many more histories than fuel regions, implying that we need to use hundreds of millions or possibly billions of particles.

There are two problems that arise in such large problems. The first is as such: the fact that we are simulating possibly hundreds of millions of particles naturally means that we will want to use hundreds of thousands of processors in parallel or else we will be waiting weeks for our results. The binary tree-based algorithms used to transmit data collectively between processors will generally scale as $\log_2 p$ where p is the number of processors. Thus, as we increase the number of processors, we also increase the amount of communication. If we run 10^7 histories per cycle in an eigenvalue calculation with 1024 processors, to broadcast 10^7 source points for the next cycle will entail sending 280 GB of data through the network assuming each source point contains 28 bytes of data! Thus, we see that for such problems, communication times may no longer be irrelevant when considering

the parallel efficiency.

The second problem is that the problem data itself may not fit on a single compute node. Let us again consider our PWR problem where we have 20 894 700 fuel regions. For each fuel region, we will have associated geometry, material, and tally data. In each material, let us also suppose we want to track 20 isotopes. For each isotope, we need to know its ZAID (4-byte integer), its cross section identifier (4-byte integer), and its number density (8-byte float). For each fuel region, we also need to tally the fission reaction rate, the (n, γ) reaction rate, the $(n, 2n)$ reaction rate, the (n, α) reaction rate, and the (n, p) reaction rate to solve the Bateman equations and determine material compositions for the next time-step. Thus, we need six pairs of 8-byte floats for each fuel region to store the tally data. When you add up the material and tally data, the total comes out to at least 496 bytes for each region. Based on this estimate, we would need about 10 GB of memory just to store the material and tally data for all the fuel regions. Add in the cross-section and geometry data and this estimate gets larger. In a typical distributed-shared memory environment where we have eight or more processors sharing memory, the SPMD parallel model requires that each process have its own copy of geometry, cross-section, material, and tally data. So for a node with eight processors, our PWR model will need at the bare minimum 80 GB of memory. Even without the addition of radial zones in each fuel pin as we have done here, one would still need 28 GB of memory. Indeed, Kelly *et al.* had to turn off variance calculation for nuclide-level tallies to reduce memory usage to a point where they could run the problem at all [5].

For the purposes of the present analysis, we focus primarily on the first problem, *i.e.* that existing parallel algorithms for Monte Carlo criticality calculations do not scale past a few compute nodes for large-scale problems. In Section 2, we describe the traditional parallel algorithm used in Monte Carlo criticality calculations and present a new algorithm. In Section 3, we present a theoretical analysis of several fission bank algorithms and a summary of test cases that validate the theoretical analysis. In Section 4, we discuss load balancing, ordering of the fission bank, and fault tolerance. Finally, in Section 5, we conclude by reviewing the salient points of this work as well as potential shortcomings of the method.

2 Algorithms

2.1 Traditional Algorithm

Monte Carlo particle transport codes commonly implement a SPMD model by having one master process that controls the scheduling of work and the remaining processes wait to receive work from the master, process the work, and then send their results to the master at the end of the simulation (or a source iteration in the case of an eigenvalue calculation). This idea is illustrated in Figure 1.

Eigenvalue calculations are slightly more difficult to parallelize than fixed source calculations since it is necessary to converge on the fission source distribution and eigenvalue before tallying. In a Monte Carlo eigenvalue calculation, a finite number of neutron histories, N, are tracked through their lifetime iteratively. If fission occurs, rather than tracking the resulting fission

neutrons, the site is banked for use in the subsequent cycle. At the end of each cycle, N source sites for the next cycle must be randomly sampled from the M fission sites that were banked to ensure that the neutron population does not grow exponentially. To ensure that the results are reproducible, one must guarantee that the process by which fission sites are randomly sampled does not depend on the number of processors. What is typically done is the following [6]:

1. Each compute node *send*s its fission bank sites to a master process;
2. The master process sorts or orders [7] the fission sites based on a unique identifier;
3. The master process samples N fission sites from the ordered array of M sites; and
4. The master process *broadcast*s all the fission sites to the compute nodes.

The first and last steps of this process are the major sources of communication overhead between cycles. Since the master process must receive M fission sites from the compute nodes, the first step is necessarily serial. This step can be completed in $O(M)$ time. The *broadcast* step can benefit from parallelization through a tree-based algorithm. Despite this, the communication overhead is still considerable.

To see why this is the case, it is instructive to look at a hypothetical example. Suppose that a calculation is run with $N = 10\,000\,000$ neutrons across 64 compute nodes. On average, $M = 10\,000\,000$ fission sites will

be produced. If the data for each fission site consists of a spatial location (three 8 byte real numbers) and a unique identifier (one 4 byte integer), the memory required per site is 28 bytes. To *broadcast* 10 000 000 source sites to 64 nodes will thus require transferring 17.92 GB of data. Since each compute node does not need to keep every source site in memory, one could modify the algorithm from a *broadcast* to a *scatter*. However, for practical reasons (e.g. work self-scheduling [8]), this is normally not done in production Monte Carlo codes.

2.2 Novel Algorithm

To reduce the amount of communication required in a fission bank synchronization algorithm, it is desirable to move away from the typical master-slave algorithm to an algorithm whereby the compute nodes communicate with one another only as needed. This concept is illustrated in Figure 2.

Since the source sites for each cycle are sampled from the fission sites banked from the previous cycle, it is a common occurrence for a fission site to be banked on one compute node and sent back to the master only to get sent back to the same compute node as a source site. As a result, much of the communication inherent in the algorithm described previously is entirely unnecessary. By keeping the fission sites local, having each compute node sample fission sites, and sending sites between nodes only as needed, one can cut down on most of the communication. One algorithm to achieve this is as follows:

1. An exclusive scan is performed on the number of sites banked, and

the total number of fission bank sites is broadcasted to all compute nodes. By picturing the fission bank as one large array distributed across multiple nodes, one can see that this step enables each compute node to determine the starting index of fission bank sites in this array. Let us call the starting and ending indices on the i-th node a_i and b_i, respectively;

2. Each compute node samples sites at random from the fission bank using the same starting seed. A separate array on each compute node is created that consists of sites that were sampled local to that node, *i.e.* if the index of the sampled site is between a_i and b_i, it is set aside;

3. If any node sampled more than N/p fission sites where p is the number of compute nodes, the extra sites are put in a separate array and sent to all other compute nodes. This can be done efficiently using the *allgather* collective operation;

4. The extra sites are divided among those compute nodes that sampled fewer than N/p fission sites.

However, even this algorithm exhibits more communication than necessary since the *allgather* will send fission bank sites to nodes that don't necessarily need any extra sites.

One alternative is to replace the *allgather* with a series of *sends*. If a_i is less than iN/p, then send $iN/p - a_i$ sites to the left adjacent node. Similarly, if a_i is greater than iN/p, then receive $a_i - iN/p$ from the left adjacent node. This idea is applied to the fission bank sites at the end of each node's array

as well. If b_i is less than $(i+1)N/p$, then receive $(i+1)N/p - b_i$ sites from the right adjacent node. If b_i is greater than $(i+1)N/p$, then send $b_i - (i+1)N/p$ sites to the right adjacent node. Thus, each compute node sends/receives only two messages under normal circumstances.

The following example illustrates how this algorithm works. Let us suppose we are simulating $N = 1000$ neutrons across four compute nodes. For this example, it is instructive to look at the state of the fission bank and source bank at several points in the algorithm:

1. The beginning of a cycle where each node has N/p source sites;
2. The end of a cycle where each node has accumulated fission sites;
3. After sampling, where each node has some amount of source sites usually not equal to N/p;
4. After redistribution, each node again has N/p source sites for the next cycle;

At the end of each cycle, each compute node needs 250 fission bank sites to continue on the next cycle. Let us suppose that p_0 produces 270 fission banks sites, p_1 produces 230, p_2 produces 290, and p_3 produces 250. After each node samples from its fission bank sites, let's assume that p_0 has 260 source sites, p_1 has 215, p_2 has 280, and p_3 has 245. Note that the total number of sampled sites is 1000 as needed. For each node to have the same number of source sites, p_0 needs to send its right-most 10 sites to p_1, and p_2 needs to send its left-most 25 sites to p_1 and its right-most 5 sites to p_3. A schematic of this example is shown in Figure 3. The data local to each node

is given a different hatching, and the cross-hatched regions represent source sites that are communicated between adjacent nodes.

3 Analysis of Communication Requirements

While the prior considerations may make it readily apparent that the novel algorithm should outperform the traditional algorithm, it is instructive to look at the total communication cost of the novel algorithm relative to the traditional algorithm. This is especially so because the novel algorithm does not have a constant communication cost due to stochastic fluctuations. Let us begin by looking at the cost of communication in the traditional algorithm

3.1 Cost of Traditional Algorithm

As discussed earlier, the traditional algorithm is composed of a series of *send*s and typically a *broadcast*. To estimate the communication cost of the algorithm, we can apply a simple model that captures the essential features. In this model, we assume that the time that it takes to send a message between two nodes is given by $\alpha + (sN)\beta$, where α is the time it takes to initiate the communication (commonly called the latency), β is the transfer time per unit of data, N is the number of fission sites, and s is the size in bytes of each fission site.

The first step of the traditional algorithm is to send p messages to the master node, each of size sN/p. Thus, the total time to send these messages is

$$t_{\text{send}} = p\alpha + sN\beta. \tag{1}$$

Generally, the best parallel performance is achieved in a weak scaling scheme where the total number of histories is proportional to the number of processors. However, we see that when N is proportional to p, the time to send these messages increases proportionally with p.

Estimating the time of the *broadcast* is complicated by the fact that different MPI implementations may use different algorithms to perform collective communications. Worse yet, a single implementation may use a different algorithm depending on how many nodes are communicating and the size of the message. Using multiple algorithms allows one to minimize latency for small messages and minimize bandwidth for long messages.

We will focus here on the implementation of *broadcast* in the MPICH2 implementation [9]. For short messages, MPICH2 uses a binomial tree algorithm. In this algorithm, the root process sends the data to one node in the first step, and then in the subsequent, both the root and the other node can send the data to other nodes. Thus, it takes a total of $\lceil \log_2 p \rceil$ steps to complete the communication where $\lceil x \rceil$ is the smallest integer not less than x. The time to complete the communication is

$$t_{\text{short}} = \lceil \log_2 p \rceil \, (\alpha + sN\beta) \, . \tag{2}$$

This algorithm works well for short messages since the latency term scales logarithmically with the number of nodes. However, for long messages, an algorithm that has lower bandwidth has been proposed by Barnett et al. [10] and implemented in MPICH2. Rather than using a binomial tree, the *broadcast* is divided into a *scatter* and an *allgather*. The time to complete

the *scatter* is $\log_2 p\alpha + \frac{p-1}{p} N\beta$ using a binomial tree algorithm. The *allgather* is performed using a ring algorithm that completes in $(p-1)\alpha + \frac{p-1}{p} N\beta$. Thus, together the time to complete the broadcast is

$$t_{\text{long}} = (\log_2 p + p - 1)\,\alpha + 2\frac{p-1}{p} sN\beta. \tag{3}$$

The fission bank data will generally exceed the threshold for switching from short to long messages (typically 8 kilobytes), and thus we will use the equation for long messages. Adding Eq. 1 and 3, the total cost of the series of *send*s and the *broadcast* is

$$t_{\text{old}} = (\log_2 p + 2p - 1)\,\alpha + \frac{3p-2}{p} sN\beta. \tag{4}$$

3.2 Cost of Novel Algorithm

With the communication cost of the traditional fission bank algorithm quantified, we now proceed to discuss the communicatin cost of the proposed algorithm. Comparing the cost of communication of this algorithm with the traditional algorithm is not trivial due to fact that the cost will be a function of how many fission sites are sampled on each node. If each node samples exactly N/p sites, there will not be communication between nodes at all. However, if any one node samples more or less than N/p sites, the deviation will result in communication between logically adjacent nodes. To determine the expected deviation, one can analyze the process based on the fundamentals of the Monte Carlo process.

The steady-state neutron transport equation for a multiplying medium

can be written in the form of an eigenvalue problem [11],

$$S(\mathbf{r}) = \frac{1}{k} \int F(\mathbf{r}' \rightarrow \mathbf{r}) S(\mathbf{r}')\, d\mathbf{r}, \tag{5}$$

where,

$\mathbf{r}$ = spatial coordinates of phase space

$S(\mathbf{r})$ = source distribution defined as the expected number of neutrons born from fission per unit phase-space volume at $\mathbf{r}$

$F(\mathbf{r}' \rightarrow \mathbf{r})$ = expected number of neutrons born from fission per unit phase space volume at $\mathbf{r}$ caused by a neutron at $\mathbf{r}'$

k = eigenvalue.

The fundamental eigenvalue of Eq. (5) is known as k_{eff}, but for simplicity we will simply refer to it as k.

In a Monte Carlo criticality simulation, the power iteration method is applied iteratively to obtain stochastic realizations of the source distribution and estimates of the k-eigenvalue. Let us define $\hat{S}^{(m)}$ to be the realization of the source distribution at cycle m and $\hat{\epsilon}^{(m)}$ be the deviation from the deterministic solution arising from the stochastic nature of the tracking process. We can write the stochastic realization in terms of the fundamental source distribution and the fluctuating component as [12]

$$\hat{S}^{(m)}(\mathbf{r}) = NS(\mathbf{r}) + \sqrt{N}\hat{\epsilon}^{(m)}(\mathbf{r}), \tag{6}$$

where N is the number of particle histories per cycle. Without loss of generality, we shall drop the superscript notation indicating the cycle as it is understood that the stochastic realization is at a particular cycle. The expected value of the stochastic source distribution is simply

$$E\left[\hat{S}(\mathbf{r})\right] = NS(\mathbf{r}) \tag{7}$$

since $E\left[\hat{\epsilon}(\mathbf{r})\right] = 0$. The noise in the source distribution is due only to $\hat{\epsilon}(\mathbf{r})$ and thus the variance of the source distribution will be

$$\mathrm{Var}\left[\hat{S}(\mathbf{r})\right] = N\mathrm{Var}\left[\hat{\epsilon}(\mathbf{r})\right]. \tag{8}$$

Lastly, the stochastic and true eigenvalues can be written as integrals over all phase space of the stochastic and true source distributions, respectively, as

$$\hat{k} = \frac{1}{N}\int \hat{S}(\mathbf{r})\, d\mathbf{r} \quad \text{and} \quad k = \int S(\mathbf{r})\, d\mathbf{r}, \tag{9}$$

noting that $S(\mathbf{r})$ is $O(1)$ since the true source distribution is not a function of N (see Nease *et al.* [13] for a thorough discussion). One should note that the expected value k calculated by Monte Carlo power iteration (*i.e.* the method of successive generations) will be biased from the true fundamental eigenvalue of Eq. 5 by $O(1/N)$ [12], but we will assume henceforth that the number of particle histories per cycle is sufficiently large to neglect this bias.

With this formalism, we now have a framework within which we can determine the properties of the distribution of expected number of fission

sites. The explicit form of the source distribution can be written as

$$\hat{S}(\mathbf{r}) = \sum_{i=1}^{M} w_i \delta(\mathbf{r} - \mathbf{r}_i) \tag{10}$$

where $\mathbf{r}_i$ is the spatial location of the i-th fission site, w_i is the statistical weight of the fission site at $\mathbf{r}_i$ (i.e. the weight of the neutron entering into a fission reaction), and M is the total number of fission sites. It is clear that the total weight of the fission sites is simply the integral of the source distribution. Integrating Eq. 6 over all space, we obtain

$$\int \hat{S}(\mathbf{r})\, d\mathbf{r} = N \int S(\mathbf{r})\, d\mathbf{r} + \sqrt{N} \int \hat{\epsilon}(\mathbf{r})\, d\mathbf{r}. \tag{11}$$

Substituting the expressions for the stochastic and true eigenvalues from Eq. 9, we can relate the stochastic eigenvalue to the integral of the noise component of the source distribution as

$$N\hat{k} = Nk + \sqrt{N} \int \hat{\epsilon}(\mathbf{r})\, d\mathbf{r}. \tag{12}$$

Since the expected value of $\hat{\epsilon}$ is zero, the expected value of its integral will also be zero. We thus see that the variance of the integral of the source distribution, i.e. the variance of the total weight of fission sites produced, is directly proportional to the variance of the integral of the noise component. Let us call this term σ^2 for simplicity:

$$\text{Var}\left[\int \hat{S}(\mathbf{r})\, d\mathbf{r}\right] = N\sigma^2. \tag{13}$$

The actual value of σ^2 will depend on the physical nature of the problem, whether variance reduction techniques are employed, etc. For instance, one could surmise that for a highly scattering problem, σ^2 would be smaller than for a highly absorbing problem since more collisions will lead to a more precise estimate of the source distribution. Similarly, using implicit capture should in theory reduce the value of σ^2.

Let us now consider the case where the N total histories are divided up evenly across p compute nodes. Since each node simulates N/p histories, we can write the source distribution as

$$\hat{S}_i(\mathbf{r}) = \frac{N}{p} S(\mathbf{r}) + \sqrt{\frac{N}{p}} \hat{\epsilon}_i(\mathbf{r}) \quad \text{for} \quad i = 1, \ldots, p \tag{14}$$

Integrating over all space and simplifying, we can obtain an expression for the eigenvalue on the i-th node:

$$\hat{k}_i = k + \sqrt{\frac{p}{N}} \int \hat{\epsilon}_i(\mathbf{r}) \, d\mathbf{r}. \tag{15}$$

It is easy to show from this expression that the stochastic realization of the global eigenvalue is merely the average of these local eigenvalues:

$$\hat{k} = \frac{1}{p} \sum_{i=1}^{p} \hat{k}_i. \tag{16}$$

As was mentioned earlier, at the end of each cycle one must sample N sites from the M sites that were created. Thus, the source for the next cycle can be seen as the fission source from the current cycle divided by the stochastic realization of the eigenvalue since it is clear from Eq. 9 that $\hat{k} = M/N$.

Similarly, the number of sites sampled on each compute node that will be used for the next cycle is

$$M_i = \frac{1}{\hat{k}} \int \hat{S}_i(\mathbf{r})\, d\mathbf{r} = \frac{N}{p} \frac{\hat{k}_i}{\hat{k}}. \tag{17}$$

While we know conceptually that each compute node will under normal circumstances send two messages, many of these messages will overlap. Rather than trying to determine the actual communication cost, we will instead attempt to determine the maximum amount of data being communicated from one node to another. At any given cycle, the number of fission sites that the j-th compute node will send or receive (Λ_j) is

$$\Lambda_j = \left| \sum_{i=1}^{j} M_i - \frac{jN}{p} \right|. \tag{18}$$

Noting that jN/p is the expected value of the summation, we can write the expected value of Λ_j as the mean absolute deviation of the summation:

$$E\left[\Lambda_j\right] = E\left[\left| \sum_{i=1}^{j} M_i - \frac{jN}{p} \right|\right] = \mathrm{MD}\left[\sum_{i=1}^{j} M_i\right] \tag{19}$$

where MD indicates the mean absolute deviation of a random variable. The mean absolute deviation is an alternative measure of variability.

In order to ascertain any information about the mean deviation of M_i, we need to know the nature of its distribution. Thus far, we have said nothing of the distributions of the random variables in question. The total number of fission sites resulting from the tracking of N neutrons can be

shown to be normally distributed via the Central Limit Theorem (provided that N is sufficiently large) since the fission sites resulting from each neutron are "sampled" from independent, identically-distributed random variables. Thus, $\hat{k}$ and $\int \hat{S}(\mathbf{r})\, d\mathbf{r}$ will be normally distributed as will the individual estimates of these on each compute node.

Next, we need to know what the distribution of M_i in Eq. 17 is or, equivalently, how $\hat{k}_i/\hat{k}$ is distributed. The distribution of a ratio of random variables is not easy to calculate analytically, and it is not guaranteed that the ratio distribution is normal if the numerator and denominator are normally distributed. For example, if X is a standard normal distribution and Y is also standard normal distribution, then the ratio X/Y has the standard Cauchy distribution. The reader should be reminded that the Cauchy distribution has no defined mean or variance. That being said, Geary [14] has shown that, for the case of two normal distributions, if the denominator is unlikely to assume values less than zero, then the ratio distribution is indeed approximately normal. In our case, $\hat{k}$ absolutely cannot assume a value less than zero, so we can be reasonably assured that the distribution of M_i will be normal.

For a normal distribution with mean μ and distribution function $f(x)$, it can be shown that

$$\int_{-\infty}^{\infty} f(x)\,|x-\mu|\;dx = \sqrt{\frac{2}{\pi}\int_{-\infty}^{\infty} f(x)\,(x-\mu)^2\;dx} \tag{20}$$

by substituting the probability distribution function of a normal distribution for $f(x)$, making a change of variables, and integrating both sides. Thus the

mean absolute deviation is $\sqrt{2/\pi}$ times the standard deviation. Therefore, to evaluate the mean absolute deviation of M_i, we need to first determine its variance. Substituting Eq. 16 in Eq. 17, we can rewrite M_i solely in terms of $\hat{k}_1, \ldots, \hat{k}_p$:

$$M_i = \frac{N\hat{k}_i}{\sum_{j=1}^{p} \hat{k}_j}. \tag{21}$$

Since we know the variance of $\hat{k}_i$, we can use the error propagation law to determine the variance of M_i:

$$\mathrm{Var}\left[M_i\right] = \sum_{j=1}^{p} \left(\frac{\partial M_i}{\partial \hat{k}_j}\right)^2 \mathrm{Var}\left[\hat{k}_j\right] + \sum_{j \neq m} \sum_{m=1}^{p} \left(\frac{\partial M_i}{\partial \hat{k}_j}\right)\left(\frac{\partial M_i}{\partial \hat{k}_m}\right) \mathrm{Cov}\left[\hat{k}_j, \hat{k}_m\right] \tag{22}$$

where the partial derivatives are evaluated at $\hat{k}_j = k$. Since $\hat{k}_j$ and $\hat{k}_m$ are independent if $j \neq m$, their covariance is zero and thus the second term cancels out. Evaluating the partial derivatives, we obtain

$$\mathrm{Var}\left[M_i\right] = \left(\frac{N(p-1)}{kp^2}\right)^2 \frac{p\sigma^2}{N} + \sum_{j \neq i} \left(\frac{-N}{kp^2}\right)^2 \frac{p\sigma^2}{N} = \frac{N(p-1)}{k^2p^2}\sigma^2. \tag{23}$$

Through a similar analysis, one can show that the variance of $\sum_{i=1}^{j} M_i$ is

$$\mathrm{Var}\left[\sum_{i=1}^{j} M_i\right] = \frac{Nj(p-j)}{k^2p^2}\sigma^2 \tag{24}$$

Thus, the expected amount of communication on node j, i.e. the mean absolute deviation of $\sum_{i=1}^{j} M_i$ is proportional to

$$E\left[\Lambda_j\right] = \sqrt{\frac{2Nj(p-j)\sigma^2}{\pi k^2 p^2}}. \tag{25}$$

This formula has all the properties that one would expect based on intuition:

- As the number of histories increases, the communication cost on each node increases as well;
- If $p = 1$, i.e. if the problem is run on only one compute node, the variance will be zero. This reflects the fact that exactly N sites will be sampled if there is only one node.
- For $j = p$, the variance will be zero. Again, this says that when you sum the number of sites from each node, you will get exactly N sites.

We can determine the node that has the highest communication cost by differentiating Eq. 25 with respect to j, setting it equal to zero, and solving for j. Doing so yields $j_{\max} = p/2$. Interestingly, substituting $j = p/2$ in Eq. 25 shows us that the maximum communication cost is actually independent of the number of nodes:

$$E\left[\Lambda_{j_{\max}}\right] = \sqrt{\frac{N\sigma^2}{2\pi k^2}}. \tag{26}$$

3.3 Validation

To ensure that any assumptions made in the foregoing analysis are sound, several test cases were run to compare experimental results with the theoretical analysis. The algorithm described in section 2.2 was implemented in a simple Monte Carlo code [15]. The number of compute nodes and histories for each case are shown in Table I. Each case was run for 10 000 cycles and at the end of each run, the standard deviation of the number of fission bank

sites sent to neighboring nodes was determined for each node as a proxy for the communication cost. For Case 1, the data was fit to the following function (with the same dependence on j and p as in Eq. 25) using a least squares regression:

$$f(j, p, \beta) = \frac{\beta}{p}\sqrt{j(p-j)} \tag{27}$$

The fitting coefficient β was then used to predict the expected amount of communication for the other three cases. Figure 4 shows the expected number of fission bank sites sent or received from neighboring compute nodes, $E[\Lambda_j]$, for Cases 1 and 2 along with the least squares regression fit based on Eq. 27 for Case 1 and the prediction for Case 2. Figure 5 shows the expected number of fission bank sites sent or received from neighboring compute nodes for Cases 3 and 4 along with the predicted fits.

A few observations can be made from these figures. Firstly, the data from the four test cases demonstrates that the foregoing theoretical analysis is indeed correct. Furthermore, one can observe from these two figures that the maximum communication does occur for node $j_{\max} = p/2$ and that this maximum is indeed independent of p as predicted.

It is also instructive to check our assumption that $\sum_{i=1}^{j} M_i$ is normally distributed. One way of doing this is through a quantile-quantile (Q-Q) plot, comparing the observed quantiles of the data with the theoretical quantiles of the normal distribution. If the data are normally distributed, the points on the Q-Q plot should lie along a line. Figure 6 shows a Q-Q plot for the number of fission bank sites sent or received on the first node for the Case 1. The data clearly lie along a straight line and thus the data are normally

distributed. This conclusion was also confirmed using the Shapiro-Wilk test for normality [16].

3.4 Comparison of Communication Costs

In section 3.1, the communication time of the traditional fission bank algorithm was estimated using a simple model for the transfer time based on latency and bandwidth. The same can be done for the novel algorithm. As described earlier, each node should send or receive only two messages, one to each of its neighbors. Moreover, we concluded that the maximum amount of data will be transferred for node $j_{\max} = p/2$. Thus, we can estimate the communication time for the novel algorithm as

$$t_{\text{new}} = 2\alpha + s\sqrt{\frac{2N\sigma^2}{\pi k^2}}\beta \tag{28}$$

To compare the communication time of the two algorithms, let us assume that the size of the data is large enough that the latency is negligible. The ratio of communication times is

$$\frac{t_{\text{old}}}{t_{\text{new}}} = \frac{\left(\log_2 p + 2p - 1\right)\alpha + \frac{3p-2}{p} sN\beta}{2\alpha + s\sqrt{\frac{2N\sigma^2}{\pi k^2}}\beta} \approx \frac{(3p-2)\,k\sqrt{N\pi/2}}{p\sigma}. \tag{29}$$

In the limit of large p, this ratio becomes

$$\lim_{p\to\infty} \frac{t_{\text{old}}}{t_{\text{new}}} = \sqrt{\frac{N\pi}{2}} \cdot \frac{3k}{\sigma}. \tag{30}$$

To fully appreciate what this means, we can infer values of σ and k from the aforementioned test cases to determine what the ratio actually is. Recall that σ tells us how much stochastic noise there is in the integrated fission source. The sample standard deviation for our validation cases was $\sigma = 1.73$. Using this value and $k = 1$, we estimate that the novel algorithm would be about 2000 times faster than the traditional algorithm for $N = 1\,000\,000$. Also note that as N increases, the novel algorithm will outperform the traditional algorithm by an even wider margin.

To determine the actual performance of the novel algorithm relative to the traditional algorithm, both algorithms were implemented in the aforementioned simple Monte Carlo code. A separate simulation using each of the algorithms was run from a single compute node up to 88 compute nodes in parallel. In each case, the number of histories was 40 000 times the number of compute nodes so that each compute node simulates the same number of histories. Each simulation was run for 20 cycles. Figure 7 shows the total time spent on fission bank synchronization for the traditional and novel algorithms. We see that the performance of the novel algorithm doesn't quite meet the predicted performance based on the above analysis, but this is not surprising given the number of simplifying assumptions made in the course of the analysis. Most importantly, we have ignored the time spent sampling fission sites and copying data in memory which may become non-negligible for large N. Notwithstanding, the new algorithm performs nearly two orders of magnitude faster than the traditional algorithm for large p and large N.

4 Other Considerations

A discussion of any parallel algorithm for Monte Carlo codes would not be complete without considering how to achieve high parallel efficiency on heterogeneous architectures and how to maintain reproducibility of results. In this section, we will discuss how load balancing requirements may affect the proposed algorithm and reproducibility as well as briefly mentioning the impact of fault tolerance.

4.1 Load Balancing

One important requirement for a parallel Monte Carlo calculation is proper load-balancing, *i.e.* it is undesirable to have a compute node sitting idle with no work to do while other compute nodes are still busy working. This is especially the case when the hardware architecture is heterogeneous (having different types of processors in a single cluster). In the traditional parallel algorithm, work self-scheduling is achieved by having each slave node request small batches of work from the master, and as each batch is completed, the slave may request more work. By breaking up the problem into smaller batches, this ensures that in a single source iteration, a processor that is twice as fast as another processor will also be assigned twice as many histories to compute, and thus all processors should finish their work at approximately the same time (assuming that the time to complete a single history is constant irrespective of its properties).

The novel algorithm we have presented here precludes the use of the aforementioned self-scheduling algorithm since an important aspect of the

algorithm is to assign histories sequentially to the nodes to preserve their order. It should be noted that if one did not care to preserve reproducibility in a calculation, the self-scheduling scheme could easily be applied for load-balancing with the new fission bank algorithm.

When used on a homogeneous cluster, we don't foresee a great loss in efficiency due to the lack of load balancing in the novel algorithm. Studies using MCNP on a homogeneous Linux cluster have shown a 5-10% loss in parallel efficiency when not using any sort of load balancing [8].

It is also possible to employ a basic means of load balancing for heterogeneous architectures in the fission bank algorithm presented here by "tuning" the algorithm to the specific characteristics of the cluster. If one were to measure the performance of each type of processor on the cluster in terms of particles processed per second, the number of histories on each processor could be adjusted accordingly instead of merely distributing particles uniformly (N/p on each processor).

4.2 Ordering of the Fission Bank

The order of the fission sites produced at the end of a source iteration will depend on the order in which the particle histories were processed. Thus, if a calculation is run in parallel using the master-slave load balancing scheme described above, the order of the fission sites is not guaranteed to be reproducible since the order in which the particle histories are simulated will depend on the actions of independent processors. It is thus necessary to sort or re-order the fission sites since the sampling of source sites depends on this ordering in order to obtain a reproducible result.

A traditional sorting of the fission bank would be done using a standard quicksort that can be completed in $O(N \log N)$ steps. Brown and Sutton developed a method of re-ordering the fission sites that can be done in $O(N)$ steps [7]. Anecdotally, we have observed that in systems using a high-speed network interconnect such as InfiniBand, the sorting of the fission bank is the major contributor to decreased parallel efficiency in fission bank synchronization.

Using the fission bank algorithm we have developed herein, the order in which particle histories are processed is always the same since we simply allocate the first set of N/p particles to the first processor, the second set of N/p particles to the second processor, etc. Thus, it is not necessary to sort or re-order the fission bank in order to preserve reproducibility of results.

4.3 Fault Tolerance

On a parallel architecture with a large number of processors, network interconnects, hard disks, and memory, it is desirable to have some means of fault tolerance to ensure that not all results are lost in the event of a hardware failure. In current Monte Carlo codes, this can be achieved by having the slave nodes periodically rendezvous and having a single processor dump data to a file that can be used to restart the run. While providing insurance against lost simulation time, performing fault tolerance this way unfortunately degrades parallel performance since it entails collective communication between all processors. Notwithstanding, the algorithm we have presented here does not inhibit the use of fault tolerance in this manner.

5 Conclusions

We have presented a new method for parallelizing the source iterations in a Monte Carlo criticality calculation. This algorithm takes advantage of the fact that many of the fission sites produced on one processor can be used as source sites on that same processor and avoids unnecessary communication between processors.

Analysis of the algorithm shows that it should outperform existing algorithms for fission bank synchronization and that furthermore, the performance gap increases for an increasing number of histories or processors. Test results on a simple 3D structured mesh Monte Carlo code confirm this finding. The analysis also shows that the maximum amount of communication in the algorithm is independent of the number of processors and instead will depend on the number of histories per cycle and the physical characteristics of the problem at hand. Again, testing using our simple Monte Carlo code confirm this prediction.

The reader should keep in mind that while the algorithm presented here will significantly improve the time necessary to sample and distribute fission sites between cycles, it has no effect on the actual transport simulation of particles moving through a geometry. Thus, it will not improve smaller simulations that would typically run on a workstation. However, for large simulations that necessitate the use of a large cluster or supercomputer to complete in a reasonable amount of time, this novel algorithm will improve the parallel efficiency and is an important step in achieving scalability up to thousands of processors.

Potential shortcomings of the present algorithm include the inability to provide load balancing using existing algorithms for heterogeneous computer architectures. A basic method to provide load balancing in such situations based on "tuning" the algorithm was suggested, although it has not be tested as of yet.

References

[1] N. METROPOLIS and S. ULAM, "The Monte Carlo Method," *J. Am. Stat. Assoc.*, **44**, 335 (1949).

[2] E. TROUBETZKOY, H. STEINBERG, and M. KALOS, "Monte Carlo Radiation Penetration Calculations on a Parallel Computer," *Trans. Am. Nucl. Soc.*, **17**, 260 (1973).

[3] F. B. BROWN and W. R. MARTIN, "Monte Carlo Methods for Radiation Transport Analysis on Vector Computers," *Prog. Nucl. Energy*, **14**, 269 (1984).

[4] J. E. HOOGENBOOM and W. R. MARTIN, "A Proposal for a Benchmark to Monitor the Performance of Detailed Monte Carlo Calculation of Power Densities in a Full Size Reactor Core," in *Proc. Int. Conf. Mathematics, Computational Methods, and Reactor Physics*, Saratoga Springs, New York, 2009.

[5] D. J. KELLY, T. M. SUTTON, T. H. TRUMBULL, and P. S. DOBREFF, "MC21 Monte Carlo Analysis of the Hoogenboom-Martin Full-Core PWR Benchmark Problem," in *Proc. PHYSOR*, Pittsburgh, Pennsylvania, 2010.

[6] X-5 Monte Carlo Team, "MCNP - A General Monte Carlo N-Particle Transport Code, Version 5," LA-UR-03-1987, Los Alamos National Laboratory (2005).

[7] F. BROWN and T. SUTTON, "Reproducibility and Monte Carlo Eigenvalue Calculations," *Trans. Am. Nucl. Soc.*, **65**, 235 (1992).

[8] F. BROWN, "Fundamentals of Monte Carlo Transport," LA-UR-05-4983, Los Alamos National Laboratory (2005).

[9] R. THAKUR, R. RABENSEIFNER, and W. GROPP, "Optimization of Collective Communication Operations in MPICH," *Int. J. High Perform. Comput. Appl.*, **19**, 119 (2005).

[10] M. BARNETT et al., "Interprocessor Collective Communication Library (InterCom)," in *Proc. Supercomputing '94*, 1994.

[11] J. LIEBEROTH, "A Monte Carlo Technique to Solve the Static Eigenvalue Problem of the Boltzmann Transport Equation," *Nukleonik*, **11**, 213 (1968).

[12] R. J. BRISSENDEN and A. R. GARLICK, "Biases in the Estimation of k_{eff} and its Error by Monte Carlo Methods," *Ann. Nucl. Energy*, **13**, 63 (1986).

[13] B. NEASE, T. UEKI, T. SUTTON, and F. BROWN, "Instructive Concepts about the Monte Carlo Fission Source Distribution," in *Proc. Int. Conf. Mathematics, Computational Methods, and Reactor Physics (M&C 2009)*, Saratoga Springs, New York, 2009.

[14] R. C. GEARY, "The Frequency Distribution of the Quotient of Two Normal Variates," *J. Roy. Stat. Soc.*, **93**, 442 (1930).

[15] P. ROMANO, F. BROWN, and B. FORGET, "Data Decomposition for Monte Carlo Transport Applications: Initial Results and Findings," LA-UR-09-05721, Los Alamos National Laboratory (2009).

[16] S. S. SHAPIRO and M. B. WILK, "An Analysis of Variance Test for Normality (Complete Samples)," *Biometrika*, **52**, 591 (1965).

Table I: Four test cases for fission bank algorithm.

Case	Compute Nodes (p)	Histories (N)
1	8	80 000
2	8	160 000
3	16	80 000
4	16	160 000

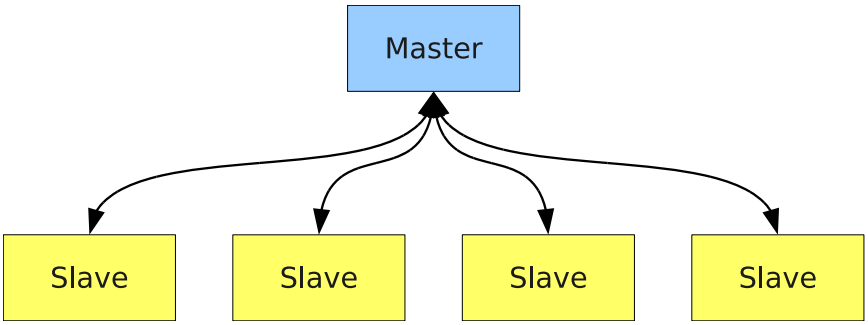

Figure 1: Typical master-slave algorithm.

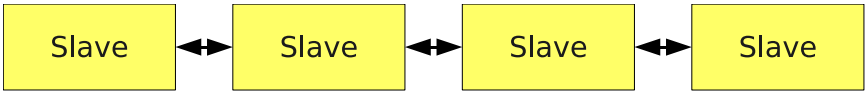

Figure 2: Desired communication topology with no master process.

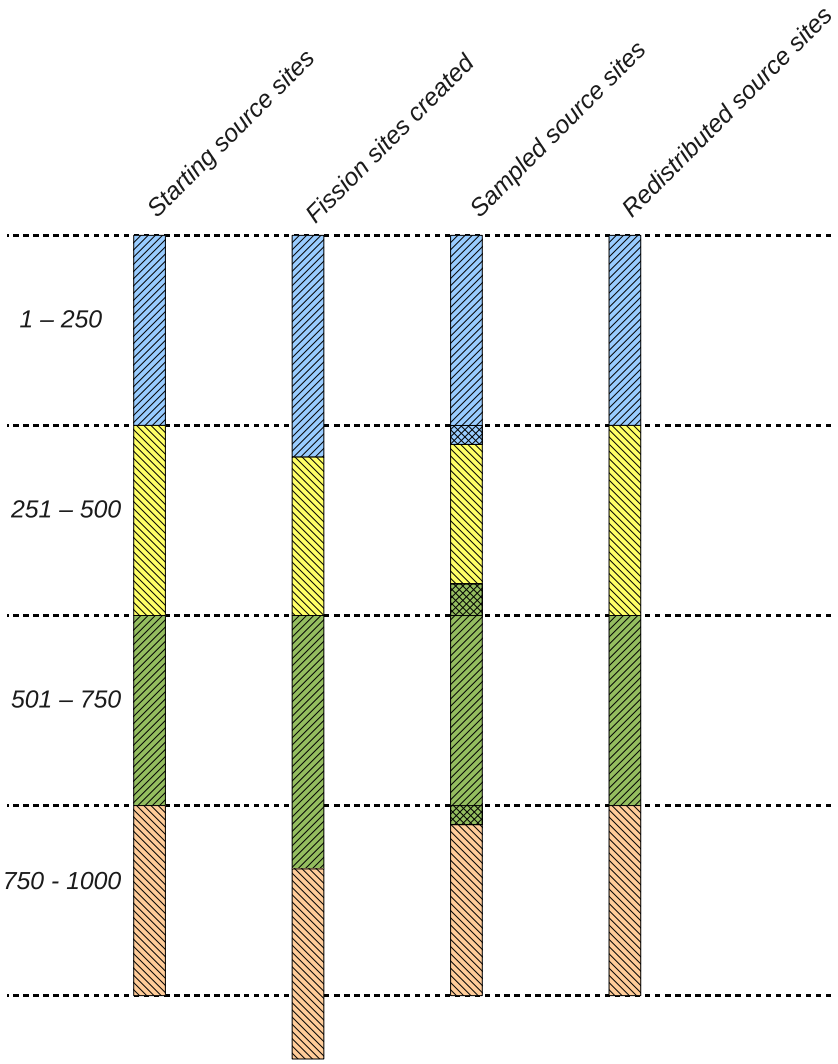

Figure 3: Example scenario illustrating new fission bank algorithm.

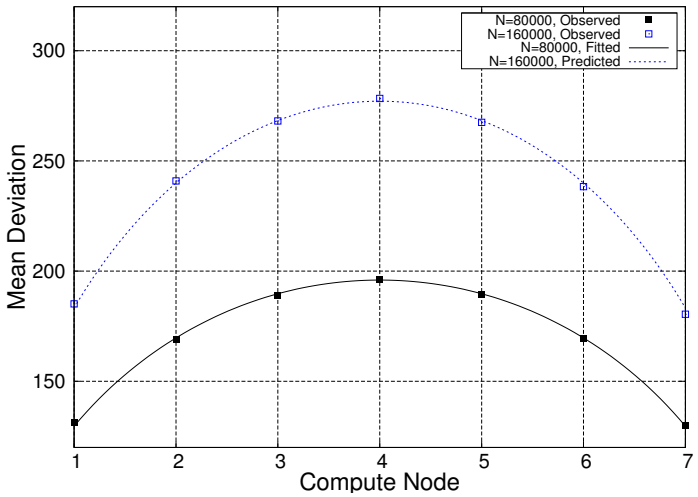

Figure 4: Expected number of fission bank sites sent to neighboring nodes using 8 compute nodes

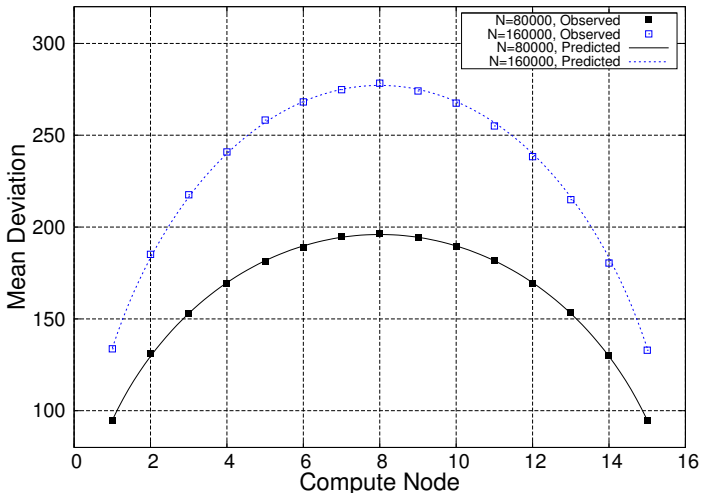

Figure 5: Expected number of fission bank sites sent to neighboring nodes using 16 compute nodes

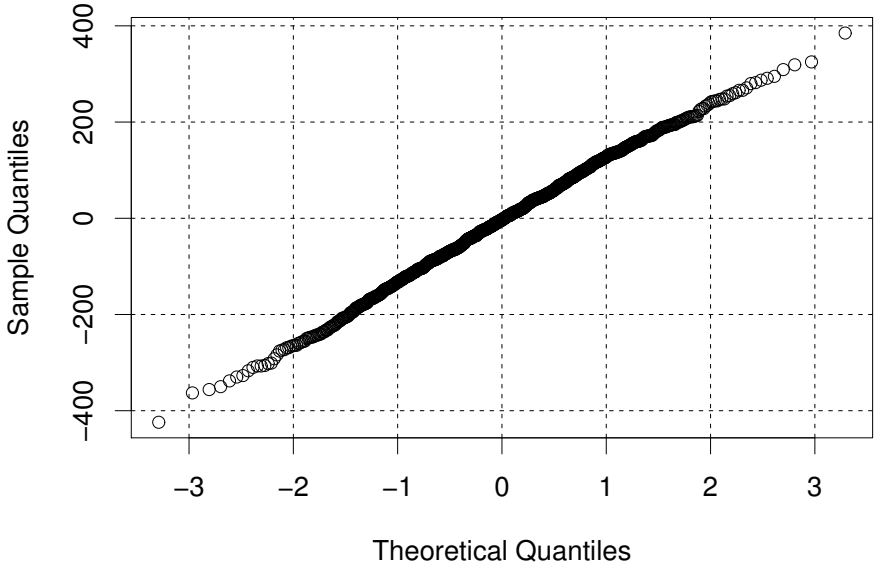

Figure 6: Q-Q plot of M_1 for Case 1

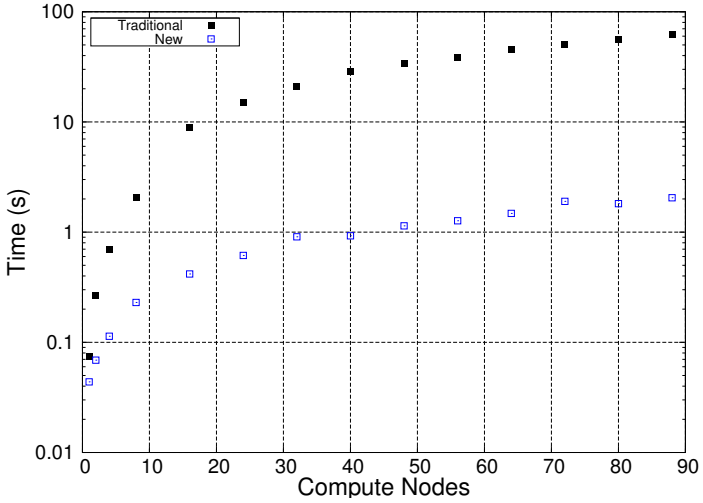

Figure 7: Execution time for fission bank algorithms