Data Decomposition in Monte Carlo Neutron Transport Simulations using Global View Arrays

Nan Dun[*,†] Hajime Fujita[*,†] John R. Tramm[†] Andrew A. Chien[*,†] Andrew R. Siegel[*,†]
[*]*University of Chicago, Department of Computer Science*
[†]*Argonne National Laboratory, Mathematics and Computer Science Division*

Abstract

Accommodating large tally data can be a challenging problem for Monte Carlo neutron transport simulations. Current approaches include either simple data replication, or are based on application-controlled decomposition such as domain partitioning or client/server models, which are limited by either memory cost or performance loss. We propose and analyze an alternative solution based on global view arrays. By using global view arrays, tallies are naturally partitioned into small globally addressable blocks that fit in the limited on-node memory of compute nodes, achieving both highly scalable memory and performance efficiency. This approach also greatly simplifies the programmability compared to application-controlled approaches. Our implementation is based on integrating a global view library built on MPI one-sided communication, Global View Resilience (GVR), into the OpenMC Monte Carlo transport code. The RMA-based global view array implementation is able to achieve a speedup of over 95× at 256 processes. Our results improve scalability significantly compared to the tally server approach and are better than any other published results, indicating that global view array is a promising alternative to enable full-core LWR analysis on current and future computer systems.

Keywords: Monte Carlo neutron transport data decomposition global array one-sided communication exascale computing

1 Introduction

Monte Carlo (MC) methods have shown a number of potential advantages over conventional deterministic methods in carrying out nuclear reactor core simulations [16, 25]: the capability of simulating arbitrary geometrical and physics complexity, no approximation for neutron energy dependence, and inherent extreme parallelism for modern HPC architectures. However, there are still two major challenges that prevent MC methods from being a realistic choice for full-core simulation. One is the enormous computational effort required to achieve acceptable statistics and source convergence, and the other is excessive demand of memory due to large cross section and tally data [16]. Current tally accumulation approaches include either simple data replication, or are based on application-controlled decomposition such as domain partitioning or client/server models, which are limited by either memory cost or performance loss. Accordingly, effective algorithms and implementations of MC simulation are still

[*]Email address: dun@cs.uchicago.edu (Nan Dun), hfujitah@cs.uchicago.edu (Hajime Fujita), jtramm@mcs.anl.gov (John R. Tramm), achien@cs.uchicago.edu (Andrew A. Chien), siegela@mcs.anl.gv (Andrew A. Siegel)

required as a matter of urgency to enable scientists to harness the power of current and future exascale systems to conduct full-scale reactor simulation.

In this technical report we specifically address the problem of memory limits on the scalability of MC transport simulations, proposing and studying an alternative data decomposition strategy for tally data based on global view arrays. In the present investigation our focus is tally accumulation. However, the techniques explored may well be adaptable to other memory-limit scaling challenges faced by Monte Carlo methods.

The contributions of this report include:

- An alternative approach to data decomposition in MC transport simulation based on applying the global view array to tally data. This approach naturally decomposes tallies into small globally addressable blocks that fit in limited on-node memory of computer nodes, achieving both highly scalable memory and performance efficiency. It also provides superior programmability for particle-based parallelization of MC simulations.

- The development of a memory cost model and a performance model for tally accumulation using global view arrays. Both models show that using global view arrays is superior to other data decomposition strategies in terms of memory and performance scalability.

- Integration of an RMA-based global view array library GVR into a MC simulation production code (OpenMC). The evaluation demonstrates that using global view array can achieve better performance and scalability than other data decomposition approaches such as explicit decomposition of nodes into tracking and tally processors [21]. We identified that data distribution and collision are two major sources of overhead when using global view array. Accordingly, a buffering scheme is proposed to alleviate the data collision problem. The RMA-based global view array implementation is able to achieve a significant speed of over 95× at 256 processes with constant memory footprint. Our results improve scalability significantly compared to the tally server approach and are better than any other published results.

The report is organized as follows. In section 2, we give a brief review of MC methods and the memory-limited scaling challenges. Section 3 elaborates the global view array approach for tally accumulation. In section 4, we develop the memory and performance models for global view array approach. The experimental methods and implementation details are described in section 5. Finally, we present and analyze the evaluation results in section 6.

2 Background

2.1 Basic Algorithms

MC methods simulate the actual neutron transport, or random movement, bounded within a predefined *geometry* space by using variables to represent a particle's state (i.e., position, direction, and energy). During the life of a particle, the probability of a specific event (i.e., scattering, absorption, and fission) and corresponding parameters (i.e., distance, energy, angle, etc.) are determined by known probability distributions called *cross sections*. These events are successively sampled from the birth of the particle until it is absorbed or leaks from the system. Meanwhile, the physical parameters are estimated by accumulating the scores from such events into sums referred to as *tallies*. For example, the total reaction rate in a volume can be estimated by summing the number of reaction events that occur within the

specified volume. The entire simulation repeats this process until enough particles have been simulated to obtain their expected behavior with sufficiently low statistical uncertainty.

In reactor simulation, MC method is used to solve k-eigenvalue problem, where the k, or k-effective, is the value that describes the criticality of the system as either sub-critical ($k < 1$), critical ($k = 1$), or super-critical ($k > 1$). To maintain the balance of the reactor (i.e., transport equation), one needs to find the eigenpair: k as the eigenvalue and the distribution of fission source sites as the eigenvector. To solve this eigenvalue problem, a *generation* of N neutrons are simulated from birth to death and then repeated for multiple successive generations. The source sites of initial generation are sampled from an initial arbitrary distribution. Then source sites of the subsequent generations are created from the *fission bank* in which neutrons newly born from fission in previous generation are stored. To obtain a converged distribution of fission source sites, a number of generations needs to be iteratively processed.

2.2 Memory Footprint

During a simulation, there are three categories of data need to be stored in memory:

- Geometry — Geometry in MC simulation is non-mesh based read-only data and is represented using *constructive solid geometry*. Complex geometry structures then can be constructed by primitives such as intersection, union, or surfaces that bound them. Furthermore, duplicate geometry structures can be reused to reduce the overall memory consumption. Therefore, the geometry data generally has moderate size and can be fully placed in local memory for optimal read performance.

- Interaction cross sections — As described in 2.1, the random interactions of particles are determined by experimentally pre-measured probability distributions. The cross sections are also read-only data and accessed randomly by each process during the simulation. The size of cross section storage depends on the energy and temperature, as well as the number of nuclides present in the system. Therefore, the actual cross section size varies significantly with the specific application. In an application with considerable temperature intervals and energy points, the cross section has been estimated to potentially exceed 100 GB [25].

- Tallies — Tally data is region-based and accumulated (i.e., fetch-and-add) data, where the region, or tally region, is the volume over which the tallies should be integrated. The size of total tally data is directly proportional to the number of physical quantities to be tallied and the number of tally regions. In a realistic reactor simulation, that tally could reach terabytes size of data [25]. Unlike geometry and cross sections, tally is write-only data and not required by particles simulation, thus it is possible to accumulate tally data in an asynchronous way.

Current parallel versions of MC methods generally require fully accessible in-memory geometry, interaction cross sections, and tally data on each process. While using data replication can be ideal to achieve perfect parallelism, it is prohibitive to store replicated cross section and tally data into the local memory of one single compute node for robust full-core analysis. Therefore, appropriate data decomposition strategies are necessary to enable parallelization of full-core simulation as well as to achieve practically useful parallel efficiency.

2.3 Decomposition Strategies

While the most scalable implementation can be done by replicating all data on all nodes, the memory cost is prohibitive for realistic full-scale simulations. Thus in practice, *domain decomposition* and *data*

decomposition are two general strategies used to tackle the problem of limited on-node memory in MC simulation.

2.3.1 Domain Decomposition

For domain decomposition, the physical space of the problem is partitioned into subdomains. Each subdomain is accordingly assigned to a separate process. Each process only stores the geometry, cross sections, and tally data associated with its subdomain. Each process then only tracks particles residing in its own subdomain. When a particle travels across the boundary of a subdomain, the particle itself is passed/sent to the process that takes charge of the target subdomain. Note that domain decomposition only helps in partitioning tally data. Cross section data is still needed on every node since cross section data is not domain based and its references during the simulation is unknown a priori.

Domain decomposition has been implemented in several MC particle transport codes [4, 7, 17, 14]. In particular, Horelik et al. [14] demonstrate an implementation scaling up to a 2.39 TB tally data distributed across 512 compute nodes. However, there are obstacles to achieving sufficient parallel efficiency. For fast particles travelling long distances before absorption, high local leakage rates (probability of a particle leaving a subdomain) may lead to significant communication overhead and load imbalances [24]. More importantly, it is also pointed out that [25]: for small subdomain size on large-scale supercomputers, the total simulation time could increase drastically because of the significant penalty due to variations in local leakage rates rather than the load imbalance from non-uniform particle densities.

2.3.2 Data Decomposition

For data decomposition, the data (including geometry, cross sections, and tallies) can be stored separately in remote nodes rather than in local memory. The data is sent and received among nodes on demand during the simulation. As a result, particle-based parallelization can be used in this scenario, where particles are not passed from one process to another. Instead, each computer process independently tracks one single particle a time from birth to death.

The data decomposition strategy can be straightforwardly applied to both cross sections and tally data. Furthermore, this approach would overcome the load imbalance problem due to the non-uniform particle densities. The communication cost introduced by data transferring is the major factor that affects the simulation performance, which is related to data granularity, physical distribution, and access patterns.

The potential for data decomposition has been discussed by Brown and Martin in 2004 [3]. Romano et al. studies the feasibility of applying remote memory access for data decomposition algorithms [19], but only addressing geometry and cross section data. Subsequently, Romano et al. [21] developed a client-server based approach called the *tally server* algorithm for tally data decomposition.

In the tally server data decomposition algorithm, overall p processes are divided into c compute processes and s tally servers. Each of the compute processes is assigned a subset of particles and conducts particle-based tracking for one particle at a time. When scores of interested events are tallied, rather than storing them in local memory, the data is sent to one of the waiting tally server processes. The tally server process does not track any particles but instead continuously receives tally scores from the compute processes and accumulates them to corresponding tally bins.

Though the tally server algorithm demonstrates its potential to enable full-core analysis with sufficient tally memory footprint, there are still several inherent shortcomings of this approach. First, this approach does not fully utilize overall local memory of all processes but relies only on tally servers' memory.

Secondly, the overall simulation performance depends on the bandwidth and ratio between compute processes and tally servers [21], which means the optimal ratio could be an architecture or configuration specific value. Third, the tally server approach does not address the decomposition of cross section data because of memory assembly complexities [21]. Finally, given that the tallies are asynchronously accumulate-only data, using send/receive semantics to communicate with servers introduces non-trivial effort to implementation and debugging, and also complicates the implementation of on-node threading parallelism.

2.4 Problem Summary

Before proceeding with the description of our proposed approach, we first summarize the problems and constraints addressed in following presentation. There are several challenging aspects of memory requirement in simulating full-core reactor problems using MC methods. From this point on, we limit the discussion to tally data accumulation and leave the potential application of global view array in cross sections as future work.

1. Memory footprint: For tallies, due to the changing of fuel composition over time, one ultimately needs to tally several reaction rates on a very fine spatial mesh. The aggregate memory footprint for this class of *depletion* calculations has been estimated at approximately 1 TB [25].

2. Performance: Since MC simulation is computation intensive and tally data is asynchronously accumulate-only, one could maximize the overlap of computation and communication to improve the overall simulation performance.

3. Scalability: In a realistic full-core simulation, there are millions of computational processes concurrently issuing tally scores. Therefore, the accumulation of tally data should be scalable, in terms of both memory cost and performance, for large number of processes.

3 Global View Approach to Tally Accumulation

3.1 Global View Models

It is our contention that programming models providing a global address space aid the development of large, complex scientific applications. In these programming models, parallel tasks communicate with each other via a logically/physically shared memory space, where synchronization primitives are provided to manage concurrency. These models can be classified further into shared memory and distributed shared memory models.

- The Shared Memory programming model is widely used on Symmetric Multiprocessor (SMP) machines where multiple identical processors/cores connect to the same shared main memory. OpenMC and Pthreads are two major paradigms to enable on-node parallelism on these SMP machines.

- The Distributed Shared Memory model, or the Partitioned Global Address Space (PGAS) model [2], provides a global memory address space as well as the control of data locality. In this model, a globally accessible memory address is logically divided into partitions and each of partitions is local to different process. This design is to simplify the data referencing programmability as in

shared memory model and also preserve the data locality for performance reasons. PGAS based languages/runtimes include Unified Parallel C, Coarray Fortran, Fortress, Chapel, X10, and Global Arrays [2].

Nevertheless, the main, important characteristic of all of these models is a global address space, enabling uniform naming and load balance. In particular, uniform naming supports distributed data structures and enables one-sided access improved data referencing performance.

3.2 One-Sided Communication

One-sided communication differentiates itself from two-sided communications semantically and operationally by separating data movement and synchronizations. By one-sided communications, processes are able to directly manipulate remote data on other processes without interfering remote CPUs. The one-sided communication is especially useful for sending asynchronous small messages because the communication completion can be deferred. It also improves the programmability for global address model.

MPI-3 one-sided interfaces provide a mechanism to directly access remote memory, which allows low-latency, high-throughput, and CPU-bypassing communication. The one-side communication reduces the send and receive matching cost in message-passing interface in MPI-1. To enable remote memory access in MPI-3, each process exposes a region of its local memory, called as *window*, to others through MPI_Win_create() collective operation. Then each process is able to directly access the windows at all processes via non-blocking operations such as MPI_Put(), MPI_Get(), and MPI_Accumulate(). These calls, different from two-sided communication, specify the remote address, datatype, and other parameters to conduct a direct remote access without involving target side. The completion of one-sided communication calls is managed by synchronization calls such as MPI_Win_flush() and MPI_Win_fence(). We refer interested users to read MPI-3 standard [18].

3.3 Apply Global View to Tally Accumulation

We study the impact of global view on MC in the context of a particular global view model, Global View Resilience (GVR) [1, 8], designed for introducing computation resilience on a foundation of reliable data structures. By exploiting a global view data model, GVR provides application-driven reliability for individual data structures to allow programmer to trade off between data importance and recovery cost. One distinguished feature of GVR is to create a series of *versioned data* to reflect the transformation of computation [15, 5]. GVR is implemented on top of MPI-3, which enables utilization of one-sided communications.

In the GVR model, using global view arrays to store tally data is straightforward. Related tally bins can be organized and stored as multidimensional global arrays where each tally bin (i.e., array element) is indexed by tally criteria, such as, nuclide, tally filters and scoring functions. A tally filter specifies what events should be scored to a given tally, such as a specific mesh or an interested energy range. The scoring functions are the actual physical quantities to be tallied, such as flux, reaction rates, etc.

The global view model particularly provides following advantages for tally accumulation in MC simulations,

- *Data decomposition*: By federating on-node memory of all compute nodes, large tally data can be partitioned and properly stored in global arrays. For example, suppose 10 TB tally memory is

required in a realistic simulation, then only 1 GB of local memory is necessary for each process when run on 10 K processes, which is feasible for most modern supercomputer systems. One factor that may affect the performance of MC simulation is the data distribution in global arrays. Since the accumulation of tally occurs at random positions, we simply use block distribution to decompose the tally data.

- *Atomic and asynchronous accumulation*: Since tally data is not needed for deciding particles' random walk and tally array is globally addressable by all processes, each process can asynchronously accumulate the tally scores to corresponding tally bins specified by array indices. By utilizing atomic fetch-and-add in one-sided primitives, concurrent tally accumulations can be conducted without explicit locking.

4 Analysis

We develop a new memory and performance model for global view-based tally accumulation, and compare it to an extension of prior models for the tally server approach [21]. Following the descriptive convention in previous work [21], let p denote the total number of processes, c is the number of compute processes, and s is the number of tally servers, assuming one-to-one correspondence between processes and processor cores. Let T denote the tally data size required by a simulation problem, m as the average size of available memory on each machine node, and M as the actual tally memory cost for parallelizing the simulation. Note that the size of tally data is *not* a function of p, it is reasonable to assume T is a fixed value for one specific simulation.

Before deriving the models, we first use Figure 1 to illustrate the memory requirements of three accumulation schemes. With local accumulation, tally data needs to be replicated on all p computer process, resulting a total size pT of memory. Since we already know $T >> m$ for a realistic simulation, $M_{local} = pT$ is apparently a prohibitive size to conduct a simulation. In the global view case, the tally data is partitioned and distributed on all p processes. In the tally server scheme, the tally data is partitioned and distributed on s tally server processes. We derive the detailed memory models of tally server and global view in following sections.

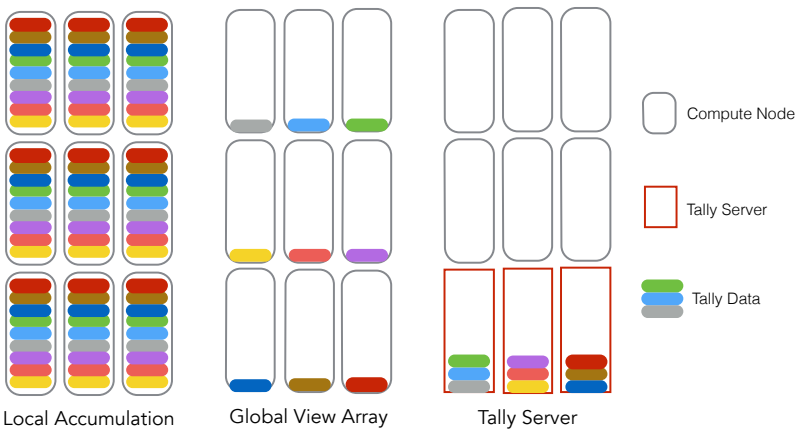

Figure 1: Illustration of memory replication and distribution in local accumulation, global view array, and tally server.

4.1 Tally Server Memory Model

In the tally server approach, the total memory cost is

$$M_{ts} = max(T, m \cdot s). \tag{1}$$

In practice, s is determined by the compute rate required to support tallies workload (e.g., $c/s = 3$ or $c/s = 7$, otherwise tally server becomes a bottleneck of communication) in a simulation, *not* the necessary memory capacity require for T [21]. As a result of $s \propto p$, the memory overhead $M_{ts} = ms$ actually grows with processes size p.

4.2 Tally Server Performance model

In previous work, Romano, et al. [21] developed and validated a performance model for the tally server approach. In each batch, a total of p processes is divided into c compute processes and s tally servers. The compute time of simulating N particles on c processes, with ideal data locality (i.e., all data is replicated and written locally on all nodes) is denoted by t_0, while the execution time by using tally server is t. Here we only discuss the case using non-blocking communication.

The time to simulate a particle is given by a distribution with a known mean μ. When the actual μ is known by conducting a simulation with specific hardware and software configurations, the overall time to complete a batch of N particles is $N\mu$. Then the ideal execution time with perfect scaling within a single batch is

$$t_0 = \frac{N\mu}{c}. \tag{2}$$

Using f as the expected number of scoring events per particle, α as the communication latency, β as the reverse bandwidth, and d as the message size, the communication time is then

$$t_{comm} = max\left(\frac{fN}{c}(\alpha + d\beta), \frac{fN}{s}(\alpha + d\beta)\right). \tag{3}$$

In practice, usually $c > s$, so t_{comm} will always be the second term $fN(\alpha + d\beta)/s$ which is determined by s. Therefore, for non-blocking communication, the complete time of N particles simulation is

$$t = max(t_0, t_{comm}), \tag{4}$$

or

$$t = max\left(\frac{N\mu}{c}, \frac{fN}{s}(\alpha + d\beta)\right). \tag{5}$$

Thus, the condition for hiding the communication is

$$\mu \geq \frac{c}{s} f(\alpha + d\beta). \tag{6}$$

4.3 Global View Memory model

In the global view model, the tally data is evenly distributed on all p processes. Therefore, the total memory overhead is

$$M_{gv} = T/N \times N = T, \tag{7}$$

where each node hosts a fraction/block of tally data as $b = T/N$.

From Equation 1 and Equation 7, we are able to estimate the memory cost by using different accumulation schemes. For example, we assume a realistic simulation problem produces 1 TB tally data, and each process has 2 GB memory capacity for tally data. Figure 2 illustrates the analytical memory cost growth for tallies as a function of the number of processes. For tally server approach, the simulation can be conducted from 1 K processes when $c/s = 1$, but its memory cost grows with a coefficient 1 with the number of processes. For $c/s = 3$ and $c/s = 7$, they show a moderate growth of memory but require more initial processes.

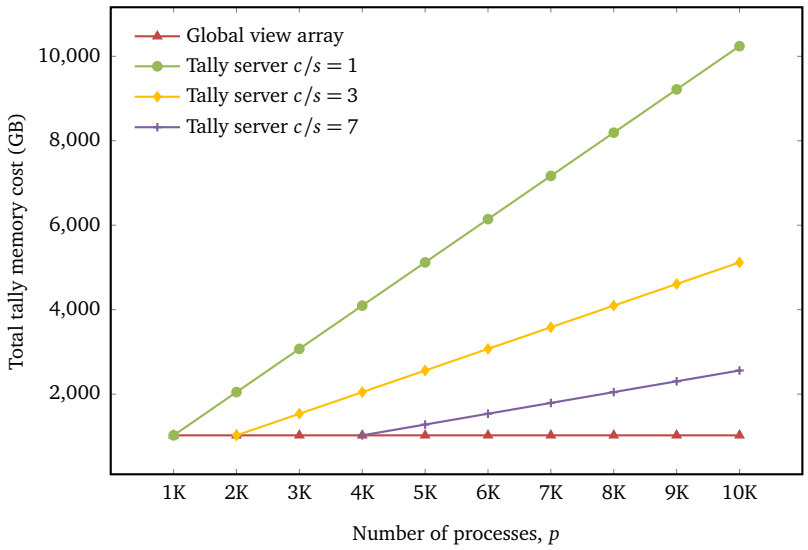

Figure 2: Analytical memory cost growth for tallies versus machine node count.

4.4 Global View Performance model

Similar as in section 4.2, the simulation limit is based on N particles, but instead using p nodes. Rendering the ideal execution time within a single batch is

$$t_0 = \frac{N\mu}{p}. \tag{8}$$

Similarly, the non-blocking one-sided communication time is

$$t_{comm} = max(T_{sends}, T_{recvs}) = max\left(\frac{fN}{p}(\alpha + d\beta) + \frac{fN}{p}(\alpha + d\beta)\right) = \frac{fN}{p}(\alpha + d\beta). \tag{9}$$

Therefore, by non-blocking communication, the complete time of N particles simulation is

$$t = max\left(\frac{N\mu}{p}, \frac{fN}{p}(\alpha + d\beta)\right). \tag{10}$$

Thus, for global view approach, the condition for hiding the communication is

$$\mu \geq f(\alpha + d\beta). \tag{11}$$

Comparing to Equation 6 for tally server model, the global view approach will exhibit better performance with a speedup s/c for given μ, when $c/s > 1$, which is always true for any reasonable performance configuration in practice. In fact, we want to maximize $c/s > 1$, as directly related to the application compute efficiency.

5 Methods

5.1 Software

As a workload, we use a production MC code, OpenMC [20]. It is capable of simulating 3D models based on constructive solid geometry with second-order surfaces. It was originally developed by members of the Computational Reactor Physics Group at the MIT in 2011. The application is written in FORTRAN, with support for a hybrid MPI/OpenMP parallelism. OpenMC is an open source software project available online, with contribution from various universities, laboratories, and other organizations.

GVR is implemented as a userspace library which provides interfaces to create and manipulate globally shared array. It is built on top of MPI-3 to leverage relaxed locking semantics in MPI-3 RMA. To create a global view array, GVR library first make an MPI window to expose each process's local memory buffer to other processes. It then constructs a mapping between array index and target rank with buffer offset so that distributed memory buffers can be uniformly addressed by a single index. The overhead of using GVR is low because it plays only as an index translation layer in terms of data access. The global array implementation is based on OpenMC 0.5.2 and GVR 0.8.1, and the tally server implementation is based on OpenMC 0.5.1.

5.2 Computing Platform

Our experimental platform, the Midway cluster at the University of Chicago, consists of 284 compute nodes. Each node has dual 8-core Intel 2.6 GHz Xeon E5-2670 processors and 32 GB DDR3 1600 MHz main memory, where all nodes are linked by fully non-blocking FDR-10 Infiniband interconnect. We used MVAPICH2 2.0b compiled by GCC 4.8.2 as the underlying MPI implementation.

In our experiments, we used 32 compute nodes and assigned processes evenly on them, i.e., running one process per node node when the number of processes $p <= 32$ and $p/32$ processes per node when $p > 32$. To avoid resource contention, only one process is allowed to run on one single core. We did not bind processes to specific cores but rather left the CPU mapping to the scheduler.

5.3 Tally Implementation

Integrating GVR arrays requires fairly small changes (less than 1% LOC) of OpenMC code. During initialization, each user-defined tally is allocated separately as a 2D global array in dimension of the number of filter bins by the number of scoring functions. In particle tracking routines, the only modification is to replace the original tally scoring function with GVR array accumulation calls when a scoring event occurs. When a batch of particles simulation finishes, the variances of tally score bins can be calculated by referencing corresponding tally array. Algorithm 1 shows the pseudocode of adding global arrays in OpenMC.

Algorithm 1 Comparison of local accumulation, global view, and tally server algorithms

```
Local accumulation: Create a duplicate of entire tally arrays on each node
Global view: Create global arrays for tallies
for i ← 1 to M do
    for j ← 1 to N/p do
        while Particle j is alive do
            Process next event
            if Event satisfies filter criteria then
                Create buffer for scores
                for all Scoring functions do
                    Calculate score
                    Accumulate score to local array
                    Accumulate score to global view array
                    Add score to buffer
                end for
                Determine server destination
                Send buffer to server
            end if
        end while
    end for
    Reduce all tally duplicates to one copy
    Send 'finish' message to server
    Flush outstanding accumulate operations
end for
Write tally results to state point file
```

5.4 Monte Carlo Workloads

Unless otherwise specified, the Monte Carlo Performance Benchmark [13] model was simulated with a $289 \times 289 \times 100$ mesh for a total of 8,352,100 tally bins. The number of particles per generation is $5,000 \times p$, where p is the number of simulation processes, with 10 inactive batches (warm up without tallying) and 10 active batches (with tallying). For each test, we present the average values from five identical runs, given that the variances are considerably small. Implications of the choice of test case for our conclusions are discussed in the following section.

6 Evaluation

6.1 Memory Footprint

We begin with an analysis of memory consumption in OpenMC to formulate a boundary for following performance discussions. A total of 8,352,100 tally bins requires 191 MB (24 bytes per tally bin) memory for every combination of tally score and nuclide. Cross sections use about 500 MB when employing a unionized energy grid. In general, the cross section and tally data consume over 95% of total memory.

Figure 3 shows the total memory cost for tally data during the runtime (i.e., VmRSS) by using local memory accumulation and RMA global view arrays, respectively. In local accumulation, the tally data size increases linearly with the number of processes, while it remains constant when using the global view array. For the tally server approach, the total on-node memory consumption scales with the number of processes. Comparing to the analysis for large scale simulation presented in Figure 2, only the global view approach demonstrates good scalability in memory cost.

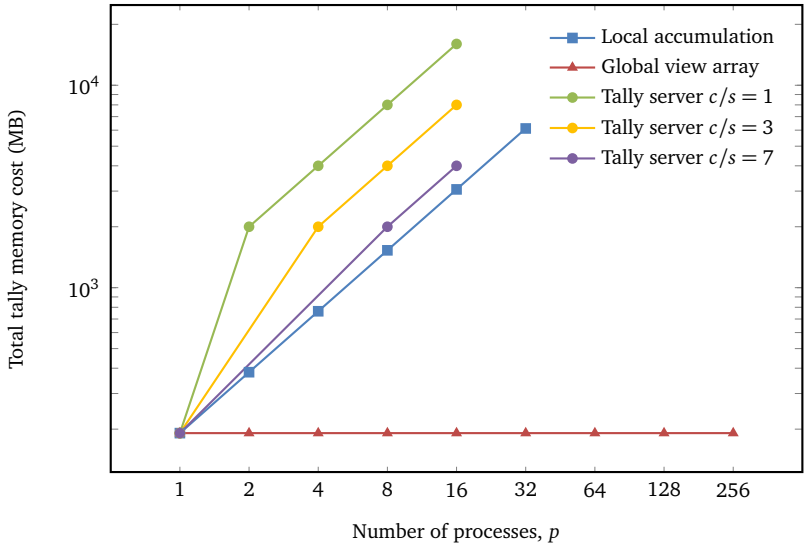

Figure 3: Empirical Memory usage for tally accumulation as the function of number of nodes.

6.2 Performance

The parallel performance and scalability of OpenMC by using the tally server approach with various c/s ratios is shown in Figure 4. First, the configuration of using $c/s = 1$ delivers best scalability up to 256

processes. The configurations of $c/s = 3$ shows better performance until p is less than 32 but saturated at $p = 4$, which the configuration using $c/s = 7$ obviously does not provide sufficient capacity for compute processes with our given benchmark settings.

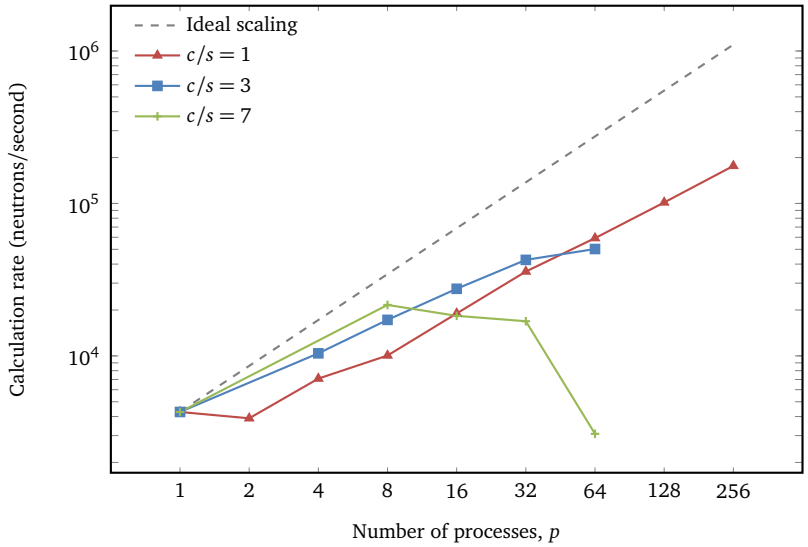

Figure 4: OpenMC performance using tally servers versus c/s ratios

By using the global view array approach, OpenMC performance has a 75% to 20% parallel efficiency from 2 to 256 processes (Fig. Figure 5). Generally, using global view arrays offers over 50% better performance than using tally server with $c/s = 1$. The major performance loss is due to half of processes are used as tally server processes. For 256 processes, the global view approach achieves a 52× speedup comparing to 41× speedup by using the tally server approach with $c/s = 1$. Note however that $c/s = 1$ has the worst memory efficiency according to Figure 2.

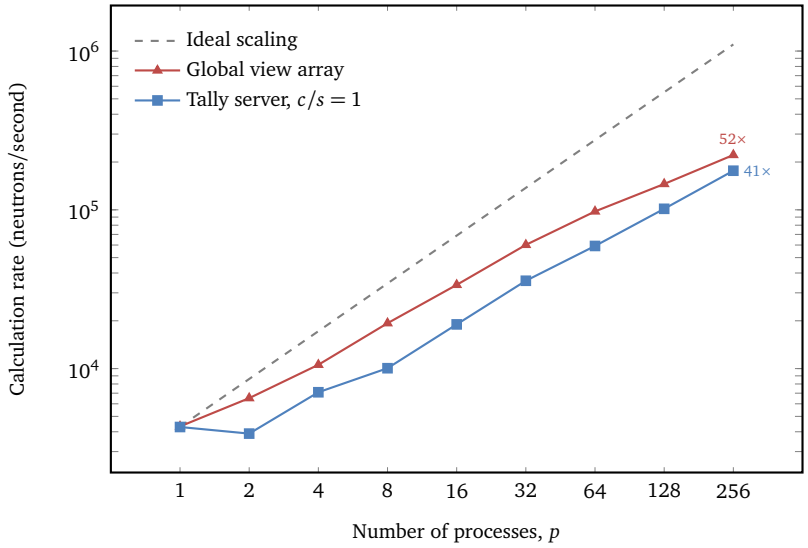

Figure 5: Global view versus tally server performance with optimal c/s ratios

We note that, even though our experiments use a smaller problem size than what is required for a full-core simulation, the results are still valid and can be applied to much larger problems with a few considerations. First, In larger problems, more nuclides appear in the reactor fuel composition, which significantly slows down the computation of macroscopic neutron cross sections and subsequently reduces the frequency of tally events. As a result, the increment of μ is significantly larger than the increment of f in Equation 11, leading to more overlapping of communication and computation. Furthermore, when the machine size grows, one would expect increasing remote access overhead due to the data distribution in global view arrays. This overhead will cause the degradation of parallel efficiency in large machines as discussed in next section.

6.3 Performance Improvement

Analytical suggests that we should be able to achieve close to ideal scaling. However, our initial measurements showed excellent nearly linear scaling to 32 nodes, but slight degradation beyond 32, and increasing as we scaled to 256. We explored three hypothesis for this degradation: 1) load imbalance in the tallies (over a batch), causing those tallies to become a bottleneck, 2) overhead of data distribution, causing increasing latency for remote data access among all processes, 3) collisions of tallies from different computations, causing the one-sided global view updates to be delayed.

6.3.1 Load Balance

To determine which phenomena is the cause, we did experiments which assessed the load balance for tallies. We computed histograms of the tallies accumulations per tally, and calculated the maximum, minimum, average, and variance for different configurations (i.e., weak scaling for $p = 4, 8, 16, 32$). This variance of accumulation count of each batch simulation is presented in Figure 6. Besides small and decreasing variances during the simulation, we observed that the maximum and average counts of accumulation for different p have small difference: the maximum ranges from 8 to 12 and the average is around 0.014. Figure 7 shows the histogram of tally accumulation count during the entire simulation, where the contribution from the first batch to the last one is shown as stacked with different colors. The accumulation events are evenly distributed in each batch and there is no significant burst accumulation. Thus, load balance is not the likely cause of performance scaling losses.

6.3.2 Tally Distribution and Collisions

We already confirmed our finding in section 6.3.1 that load balance is not the cause of parallel efficiency degradation. However, other factor of performance degradation, i.e., data distribution and collision, are not separated in previous experiment. Therefore, we conduct another experiment to differentiate the impact of data distribution and collision.

To this end, we developed a microbenchmark to simulate accumulations in OpenMC. This benchmark, creates a one dimensional global view array with 1M elements, using block distribution that partitions the global view array into p blocks evenly distributed on processes. Each client issues a balanced workload: 1M accumulate operations to to all elements of the array, using a distinct random permutation. We compare two configurations: 1) the distribution of array is fixed to 256 processes (i.e., $p = 256$) and we increase c, the number of clients that concurrently access this array, 2) the number of client accessing the array is fixed to 1 (i.e., $c = 1$) and we the array is distributed from 2 to 256 processes. Accordingly, configuration 1 allows we identify the collision overhead due to increasing concurrent accesses, while

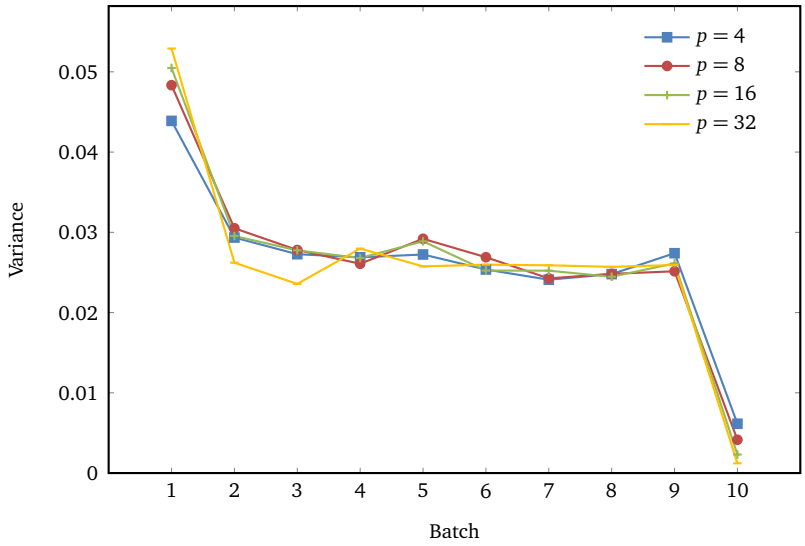

Figure 6: Variance of tally accumulation count of each batch

there is no collision in configuration 2 but only overhead due to data distribution changing. In both configurations, we also compare the sequential and random access order to see whether block access locality has potential impact. From Figure 8, we first find that sequential and random access has no significant impact on the performance. The data distribution and collision actually introduce 50% and 50% overhead respectively. For example, the elapsed time in configuration 2 increases from 0.35 sec to 7.34 sec, while elapsed time in configuration 1 increases from 7.59 sec (a base already dominated by data distribution) to 13.42 sec. Therefore, the results suggests that we can only achieve at most 1.5× speedup by reducing the collision.

Therefore, we propose a simple buffering scheme to alleviate the overhead of data collision. Note that the global view array is partitioned evenly into p blocks and distributed on p processes, resulting a one-to-one correspondence between blocks and processes. Then for each process, we allocate one buffer associated with each block in the global view array, thus p buffers in total. Instead of directly accumulating to the array, the tally scores first goes to the buffer dedicated to the target block. During each accumulation, we check whether the buffer is full. If the buffer is full, then all entries in current buffer is flushed to the corresponding block one by one at a time. Note that buffered tally scores lack sufficient locality so they cannot be combined into one accumulate operation to reduce the message size.

By using the buffering technique, we found the empirical optimal buffer size by varying buffer size and number of processes p as shown in Figure 9. Using a buffer size of 128 offers best improvement for most of cases, especially for large p, e.g., a gain of 1.4× speedup for 32 processes. The corresponding memory overhead is $8 \times 128 \times p^2$. For 256 processes, the memory cost of buffering is 64 MB in total. Since the optimal buffer size is not a function of the number of nodes, we can expect the buffer only takes a small fraction of total memory cost during the simulation.

6.4 Overall OpenMC Performance

The overall OpenMC performance is shown in Figure 10. Note that tally performance without buffering is about 30% better for global view than for tally server with best configuration (i.e., $c/s = 3$, see Figure 5).

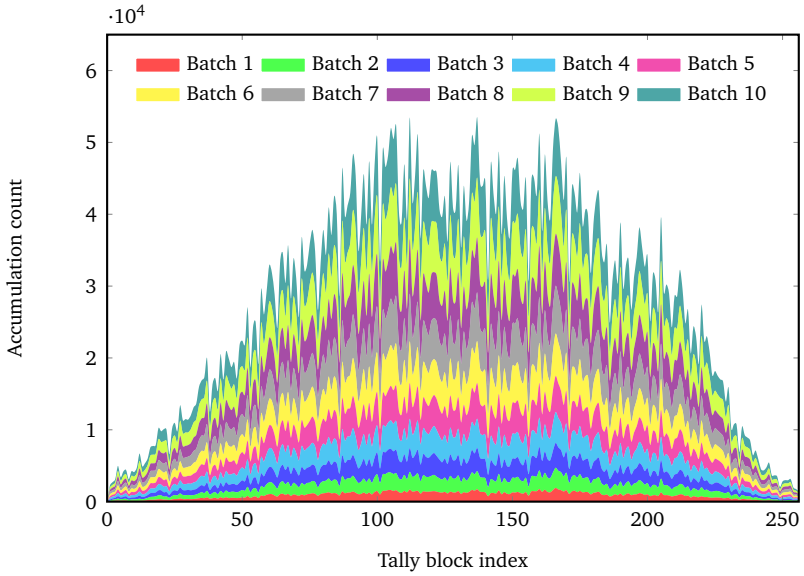

Figure 7: Histogram of tally accumulation count in OpenMC, stacked with batch statistics

For global view, scaling has a problem due to data distribution and collision (as diagnosed earlier in section 6.3), achieving only 20% of parallel efficiency relative to ideal scaling performance. By using a buffering scheme, the performance can be improved and reaches 38% of parallel efficiency relative to ideal scaling with a 98× speedup at 256 processes, recalling a speedup of 41× by the tally server approach.

We believe that the parallel efficiency can be further improved with more computation and communication overlapping by other tunings in practice. First, in larger H-M simulations where more nuclides are used, the tallying rate will significantly decrease since cross section calculation will dominate the runtime. Second, in order to achieve good performance of used MVAPICH2 implementation on the Midway system, we had to invoke MPI_Win_flush_all() once for every 128 MPI one-sided communication calls to flush all outstanding operations. Otherwise, too many outstanding requests will lead to a dramatic Infiniband performance drop in this environment. This machine-specific tuning in the GVR library helps the performance but also makes non-blocking one-sided operations effectively blocking. Therefore, the performance will also benefit from future improvements of MPI implementations.

7 Related Work

Due to the computation intensive nature of MC transport simulations, many parallelization approaches have been developed before OpenMC [6, 12]. However, they suffer from a degradation of parallel efficiency, especially for large numbers of processors. This is because most parallel MC codes are implemented in master-slave algorithms where master process easily becomes the bottleneck for fission source synchronization and tally reduction. OpenMC is equipped with three algorithms to solve these scalability problems: 1) a nearest-neighbor algorithm for reducing communication in fission bank synchronization, 2) a batching algorithm to reduce communication between successive realizations, 3) the tally server algorithm we discussed in section 2.3.2 for decomposing large tally data. With these improvements, OpenMC makes it feasible to harness massive parallelism provided by current and future

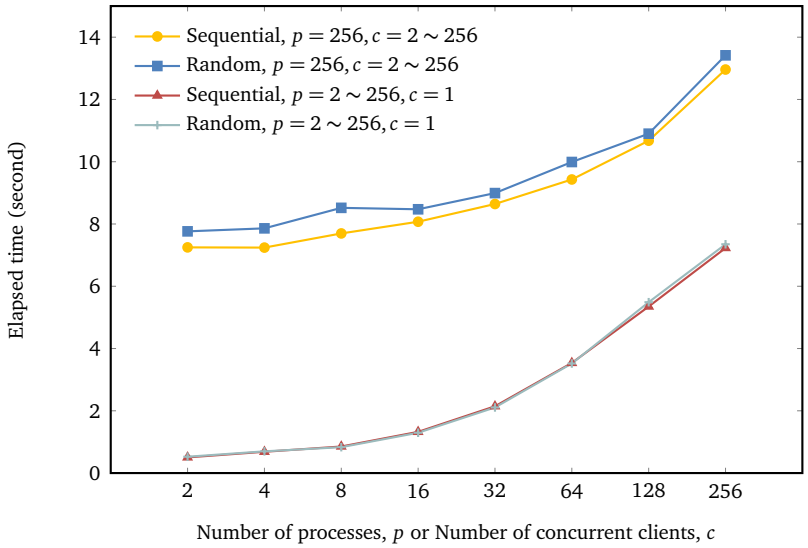

Figure 8: Effect of data distribution and collisions on RMA accumulate scaling.

exascale architectures.

Besides MC applications like OpenMC, we also demonstrate that GVR's global view approach can be also applied to a wide variety of scientific applications [9], such as NWChem [27], miniFE [11], ddcMD [26], PCG/Trilinos [10, 22], and GMRES/Trilinos [23]. With only modest code changes required, the global view scheme is able to be flexibly adapted to both match application structure and exploit application semantics.

8 Summary and Future Work

We demonstrate that global view array with RMA is a prominent alternative for tally data decomposition and accumulation in Monte Carlo particle transport simulations. Besides better expressiveness and programmability, a distributed array is highly scalable to decompose large tally data into small blocks fitting into limited on-node memory of computer nodes. By this approach, users are allowed to seamlessly shift to exascale computer systems to conduct full core analysis. The implications of our study include the effectiveness of using other RMA enabled global address space languages/libraries for data decomposition in MC transport simulations. Coupled with RMA, global view array can achieve higher performance and scalability comparing to other message-based approaches including tally server implementation.

Another advantages of using shared global arrays over the tally server algorithm is that it can transparently employs on-node threading parallelism via OpenMP. The tally server algorithm complicates the use of threading because it requires each thread to send messages to tally servers.

In future work, we will first exploit the chance to improve the throughput by reducing the overhead due to the data distribution. We will also study the effectiveness of other potential optimizations described in the tally server paper [21], such as buffering successive scoring events and topology-aware mapping. Then we will further investigate how RMA global view arrays can be applied to decompose large cross section data, where replicating/caching and online evaluation of effective cross section at arbitrary temperature are potential optimization strategies.

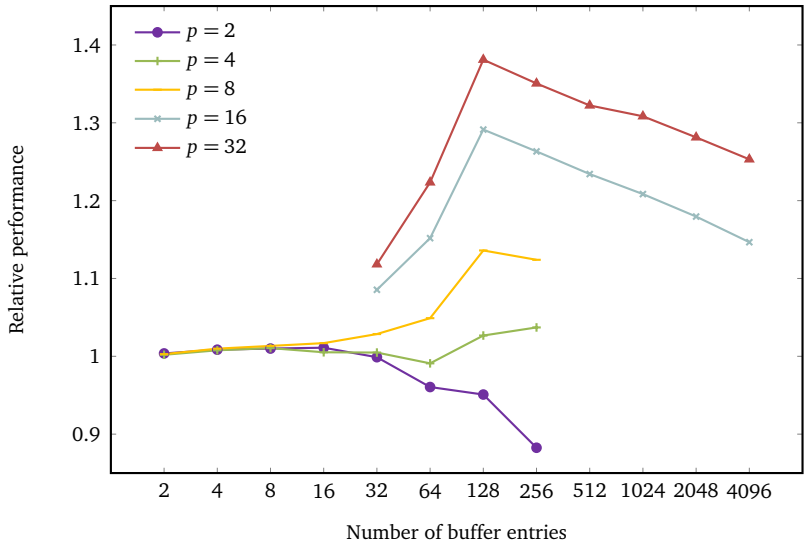

Figure 9: The impact of buffer size on the accumulate performance.

Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Award DE-SC0008603 and Contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided on Midway, high-performance computing cluster operated by the Research Computing Center at The University of Chicago.

References

[1] Global view resilience project. http://gvr.cs.uchicago.edu.

[2] Partitioned Global Address Space. http://www.pgas.org.

[3] F. B. Brown and W. R. Martin. High performance computing and Monte Carlo. *Trans. Am. Nucl. Soc.*, 91(1):279–280, 2004.

[4] T. A. Brunner and P. S. Brantley. An efficient, robust, domain-decomposition algorithm for particle Monte Carlo. *Journal of Computational Physics*, 228:3882–3890, 2009.

[5] H. Fujita, N. Dun, Z. A. Rubenstein, and A. A. Chien. Log-structured global array for efficient multi-version snapshots. In *Submitted for publication*, 2014.

[6] T. Goorley, M. James, T. Booth, F. Brown, J. Bull, L. J. Cox, J. Durkee, J. Elson, M. Fensin, R. A. Forster, J. Hendricks, H. G. Hughes, R. Johns, B. Kiedrowski, R. Martz, S. Mashnik, G. McKinney, D. Pelowitz, R. Prael, J. Sweezy, L. Waters, T. Wilcox, and T. Zukaitis. Initial MCNP5 release overview. *Nuclear Technology*, 180(3):298–315, Dec. 2012.

[7] 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

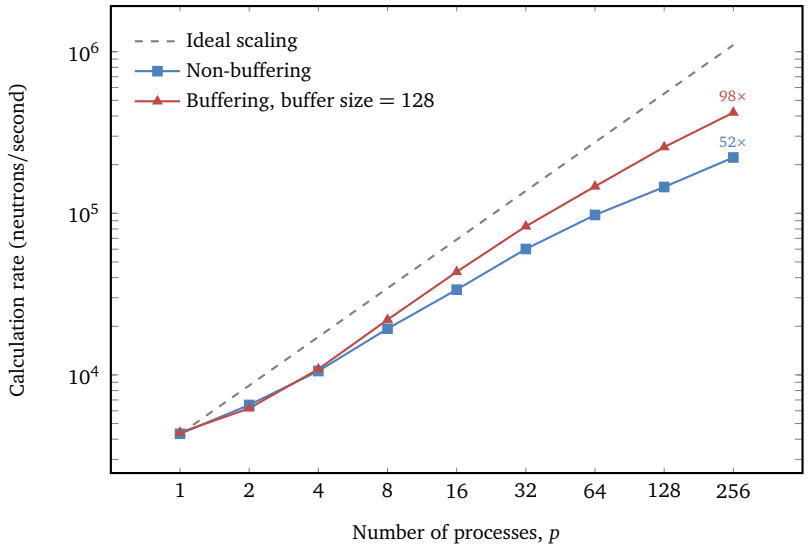

Figure 10: Performance comparison of non-buffering and buffering accumulation.

decomposition. In *International Conference on Mathematics, Computational Methods and Reactor Physics*, Saratoga Springs, New York, May 2009.

[8] GVR Team. GVR documentation, release 0.8.1-rc0. Technical Report 2014-06, University of Chicago, Department of Computer Science, 2014.

[9] GVR Team. How applications use GVR: Use cases. Technical Report 2014-05, University of Chicago, Department of Computer Science, 2014.

[10] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, and E. T. Phipps. An overview of the trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423, 2005.

[11] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving performance via mini-applications. Technical Report Tech. Rep. SAND2009-5574, Sandia National Laboratories, 2009.

[12] J. E. Hoogenboom. Is Monte Carlo embarassingly parallel? In *PHYSOR – Advances in Reactor Physics – Linking Research, Industry, and Education*, Knoxville, Tennessee, April 15–20 2012.

[13] J. E. Hoogenboom, W. R. Martin, and B. Petrovic. The Monte Carlo performance benchmark test - aims, specifications and first results. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Rio de Janeiro, Brazil, May 2011.

[14] N. Horelik, B. Forget, K. Smith, and A. Siegel. Domain decomposition and terabyte tallies with the OpenMC Monte Carlo neutron transport code. In *PHYSOR 2014 – Advances in Reactor Physics – The Role of Reactor Physics toward a Sustainable Future*, 2014.

[15] G. Lu, Z. Zheng, and A. A. Chien. When is multi-version checkpointing needed? In *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale*, FTXS '13, pages 49–56, New York, NY, USA, 2013. ACM.

[16] W. R. Martin. Challenges and prospects for whole-core Monte Carlo analysis. *J. Nucl. Eng. Technol.*, 44(2):151–160, Mar. 2012.

[17] B. T. Mervin, S. W. Mosher, T. M. Evans, J. C. Wagner, and G. I. Maldonado. Variance estimation in domain decomposed Monte Carlo eigenvalue calculations. In *PHYSOR 2012 – Advances in Reactor Physics – Linking Research, Industry, and Education*, Knoxville, Tenneessee, 2012.

[18] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 3.0*, Sept. 2012.

[19] P. Romano, B. Forget, and F. Brown. Towards scalable parallelism in Monte Carlo transport codes using remote memory access. *Prog. Nucl. Sci. Technol.*, 2:670–675, Oct. 2011.

[20] P. K. Romano and B. Forget. The OpenMC Monte Carlo particle transport code. *Ann. Nucl. Energy*, 51:274–281, 2013.

[21] P. K. Romano, A. R. Siegel, B. Forget, and K. Smith. Data decomposition of Monte Carlo particle transport simulations via tally servers. *Journal of Computational Physics*, 252:20—-36, 2013.

[22] Z. Rubenstein. Error checking and snapshot-based recovery in a preconditioned conjugate gradient solver. Master's thesis, University of Chicago, 2014.

[23] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, 1993.

[24] A. R. Siegel, K. Smith, P. Fischer, and V. Mahadevan. Analysis of communication costs for domain decomposed Monte Carlo methods in nuclear reactor analysis. *Journal of Computational Physics*, 231:3119–3125, 2012.

[25] A. R. Siegel, K. Smith, P. K. Romano, B. Forget, and K. Felker. The effect of load imbalances on the performance of Monte Carlo codes in LWR analysis. *Journal of Computational Physics*, 235:901–911, 2013.

[26] F. H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, B. R. de Supinski, J. Sexton, and J. A. Gunnels. Simulating solidification in metals at high pressure: The drive to petascale computing. *Journal of Physics: Conference Series*, 46(1):254, 2006.

[27] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. J. V. Dam, D. Wang, J. Nieplocha, E. Aprà, T. L. Windus, and W. A. deJong. NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477–1489, 2010.