

# ARRC: A random ray neutron transport code for nuclear reactor simulation

John R. Tramm<sup>a,\*</sup>, Kord S. Smith<sup>a</sup>, Benoit Forget<sup>a</sup>, Andrew R. Siegel<sup>b</sup>

<sup>a</sup>*Massachusetts Institute of Technology, Department of Nuclear Science & Engineering  
77 Massachusetts Avenue, 24-107, Cambridge, MA 02139*

<sup>b</sup>*Argonne National Laboratory, Mathematics and Computer Science Department  
9700 S Cass Ave, Argonne, IL 60439*

---

## Abstract

A massively parallel implementation of a recently developed technique for numerically integrating the transport equation, The Random Ray Method (TRRM)[1], is applied to several large reactor benchmark problems. The implementation, which is part of a new development called The Advanced Random Ray Code (ARRC), is one of the first parallel implementations of TRRM. Our goal is to better understand the accuracy and performance characteristics of TRRM on massive scale problems, and to provide community software that facilitates further algorithmic development and potentially its application to a broader class of problems. Key features of ARRC include extreme memory efficiency, domain decomposition, a task based parallel structure, and the ability to efficiently utilize Single Instruction Multiple Data (SIMD) vector units. These attributes lead to efficient performance on modern high performance computer (HPC) architectures, enabling the detailed simulation of reactor cores in three dimensions.

*Keywords:*

The Random Ray Method, Method of Characteristics, Neutron Transport, Reactor Simulation, High Performance Computing

---

## 1. Introduction

The high fidelity simulation of a full core nuclear reactor is a computational challenge well suited to the Exascale era of supercomputing due to the extremely high number of core-hours required to fully resolve a neutron transport problem[2]. A variety of transport methods are in theory capable of resolving such problems, though not all are well suited for continued increases in performance on next generation computing architectures.

We recently reported on a new, highly efficient transport method called The Random Ray Method (TRRM)[1]. TRRM can be viewed as a hybrid between Monte Carlo (MC) and Method of Characteristic (MOC) methods. It shares the basic computational structure of a multigroup MOC algorithm but with a stochastic rather than deterministic method

---

\*Corresponding author. Tel.: +1 (847) 421-1534.

Email addresses: `jtramm@mit.edu` (John R. Tramm), `kord@mit.edu` (Kord S. Smith), `bforget@mit.edu` (Benoit Forget), `siegela@mcs.anl.gov` (Andrew R. Siegel)

of discretizing the spatial and angular dependencies of the Boltzmann Neutron Transport Equation. TRRM allows for full 3D geometric flexibility similar to MC methods, requiring no assumptions of axial homogeneity in the reactor geometry to maintain computational efficiency. It also provides extreme reductions in memory usage compared to traditional MOC methods, facilitating its use for full core, high fidelity 3D reactor simulation. TRRM has also been shown to reduce algorithmic complexity compared to traditional MOC methods, allowing a sparser computational grid while maintaining accuracy. Initial work on TRRM demonstrated the new method using a test application on several benchmark problems, all 2D and 3D pin cell problems or 2D assembly scale problems of reduced complexity [1]. A complete TRRM implementation has not been reported on, and no large 3D problems have been demonstrated.

As a next step, The Advanced Random Ray Code (ARRC) has been developed as a more robust, complete implementation of TRRM targeting high fidelity full core nuclear reactor problems on next generation supercomputing systems. It is implemented in the C programming language for performance and portability on high performance computing (HPC) architectures. Three-way hybrid parallelism is achieved by using compiler directives for SIMD vectorization, OpenMP for shared memory threading, and MPI for distributed memory domain decomposition. Additionally, an improved convergence scheme using Shannon Entropy was developed and implemented in ARRC to offer more reliable stationary source detection.

The aim of this study is to analyze the accuracy and performance of ARRC to gauge its utility for large scale 3D problems. To this end, several 2D and 3D benchmark problems are executed on a variety of HPC architectures. Several challenges and their solutions relating to the usage of TRRM are also presented, including difficulties introduced by the stochastic convergence process.

## 2. Methods

### 2.1. Neutron Transport

Reactor simulation aims to calculate specific properties of the core both to make revisions to the initial reactor design and to better predict its operational behavior. Two of the most important phenomena are: 1) the eigenvalue, or criticality, of the reactor, and 2) its spatial power distribution. The eigenvalue, referred to as “k-effective”, determines the ratio of neutron populations between successive generations within the reactor. It therefore determines the balance between neutron production and loss. The power distribution within the reactor governs the thermal-hydraulic design considerations as well as the rate of burn-up of the nuclear fuel. The eigenvalue and power distribution are computed by numerically estimating the solution of the Boltzmann Neutron Transport Equation [3, pp. 111–145] [4, pp. 67–186].

There are a wide variety of common transport simulation methods and accompanying applications, such as the method of discrete ordinates (PROTEUS-SN[5], Rattlesnake[6], PARTISN[7]), Monte Carlo (MCNP[8], OpenMC[9], RMC[10], Serpent[11], TRIPOLI[12]), and the Method of Characteristics (OpenMOC[13], MPACT[14], CASMO-5[15]), among many others. The long list is due to each method or implementation often carrying its own set of strengths and weaknesses, with various applications often excelling in different problem areas and use cases. While there are many applications already, there is still ample opportunity

for new methods and applications to be useful particularly in the 3D high fidelity full core reactor simulation space. This is evidenced by the significant amount of recent research being performed in the field of high fidelity 3D MOC methods[16, 17, 18, 19].

## 2.2. The Random Ray Method

The Random Ray Method (TRRM) is a recently developed hybrid neutron transport method[1] based on the Method of Characteristics (MOC)[18, 20, 21, 22]. MOC solves a partial differential equation (PDE) by defining characteristic lines (or curves) along which the PDE is reduced to an ordinary differential equation (ODE). By solving the ODE along a set of discrete characteristic lines (sometimes in conjunction with ray tracing) and iterating on the initial conditions for the lines, a solution to the governing PDE can be numerically estimated. Traditional MOC applications use a cyclical deterministic quadrature[19, 23, 24, 25, 26] composed of many tracks to cover the phase space of the system evenly. In this way, the eigenvalue and distribution of neutrons throughout a reactor can be numerically estimated.

Unlike traditional deterministic MOC methods, TRRM does not use a cyclical deterministic quadrature. Rather, characteristic lines (also known as rays) sampled from a uniform random distribution in space and angle are followed through the system until termination criteria are met, in effect forming a stochastic integration quadrature. This cycle of sampling rays randomly and following them through the domain until they are terminated is repeated in an iterative manner, updating the scalar flux in each region of the computation after each iteration. Usage of the new method can allow for increases in accuracy due to accumulation of effective resolution over successive iterations, while also reducing computational storage requirements by removing the need to store deterministic quadrature data and the even more costly starting condition data (i.e., angular flux). While a small bias is introduced into the computation due to approximation of each ray’s starting conditions, this bias can be mitigated through usage of a “dead zone” as discussed in sections 4 and 2.4. Compared to other traditional multigroup deterministic methods, TRRM can:

- easily handle arbitrary 3D geometries[1]
- provide extreme improvements in memory efficiency[1]
- allow for a continuous angular treatment of the angular flux in the reactor, allowing for usage of continuous scattering kernels[1]
- result in significant reductions in algorithmic complexity on some simulation problems[1]
- map very well onto next generation supercomputing architectures due to its task based structure and ability to vectorize[27]

A thorough description of the method and its mathematical derivation is given by [1]. ARRC implements 3D, flat source TRRM in the form of a high performance and massively parallel application for use on modern supercomputing architectures.

### 2.3. Geometry

Fundamental to many fields of computational science is the ability to represent the geometrical structure of an object in a manner that a computer can understand. One high accuracy method that is often used in neutron transport codes is constructive solid geometry (CSG). In CSG, an object is represented exactly by definition of second order surfaces such as cylinders, spheres, and planes. Complex objects can be made by defining spaces that form the intersection, union, or complement of multiple different second order surfaces.

Many computational methods also perform “ray tracing”, wherein a virtual ray is followed through a CSG geometry to determine where it intersects, reflects, and how far it travels through each region. Traditionally, ray tracing for deterministic methods to generate a deterministic quadrature is performed once during initialization of the program and stored in memory as a table for the duration of the computation. As TRRM uses a new stochastic quadrature every iteration, the same rays will never be used twice so it is not useful to precompute and store ray tracing data. As such, TRRM method uses fully on-the-fly ray tracing, which adds to the floating point computational cost of a simulation but also reduces the memory footprint and bandwidth requirements by not storing and frequently accessing a large “tracking file”. The improved cache performance and lower bandwidth requirements resulting from on-the-fly ray tracing in TRRM have also been shown to outweigh the increased floating point costs, resulting in better overall performance compared to traditional deterministic MOC “pre-compute and store” methods as measured by the amount of time spent per angular flux integration[1]. While such performance results may vary between codes due to different ray tracing and CSG implementations as well as differing code optimization levels, it is clear at least that on-the-fly ray tracing is not prohibitively expensive. One caveat is that on-the-fly ray tracing does become relatively more computationally expensive when using very few neutron energy groups, as the amount of angular flux attenuation work that is done per ray trace is proportional to the number of energy groups. However, with high fidelity simulations often requiring on the order of 100 energy groups[28] ray tracing costs are likely to be well amortized for most high fidelity reactor computations.

### 2.4. Angular Flux Treatment

In traditional deterministic MOC codes, the angular flux spectrum at the boundary of each track is stored in memory between iterations[26]. In contrast, TRRM is able to greatly reduce the memory requirements as compared to traditional deterministic methods due to the approximations it uses to initialize angular fluxes for each ray in a simulation such as the use of a “dead zone”[1] at the beginning of all rays, wherein a new ray is tracked along its path and its angular flux updated, but no contribution to the scalar flux tally in each FSR is made. This effectively puts the beginning portion of each ray into “read only” mode, where the angular flux of the ray is continually improved without writing back to the system. After the set dead zone distance has been traversed, the ray begins behaving as normal wherein changes in the angular flux through each FSR are tallied to the scalar flux of that FSR. Importantly, these approximations mean that no angular fluxes need to be stored between power iterations as is the case with deterministic methods.

In practice, for a high fidelity MOC simulation featuring tight track spacing, a high number of azimuthal and polar angles, and a high energy group count the total storage requirements of boundary angular fluxes in deterministic MOC can be quite large. It has been shown for

example that in a high fidelity simulation of an Integral Fuel Burnable Absorber (IFBA) pincell problem with 64 energy groups, a traditional deterministic MOC method required 100.35 MB of boundary flux data and 42.75 MB of tracking file data as compared to only 0.2198 MB total data for the TRRM method[1]. Even if the tracking file was eliminated in the deterministic code in favor of performing ray tracing on-the-fly to minimize memory usage, TRRM would still be using 456x less memory. On a high fidelity IFBA assembly problem with a larger surface area to volume ratio, it was shown that the traditional deterministic MOC method required 170.53 MB of boundary flux data and 936.72 MB of tracking file data as compared to 14.69 MB of data total for the TRRM method[1]. In this case, if the tracking file were eliminated in the deterministic method in favor of on-the-fly ray tracing, TRRM would still be using at least 11.6x less memory.

It is important to note that the approximations used for starting angular flux conditions in TRRM leverage the stochastic nature of the algorithm to ensure minimal biases are introduced. The techniques used in TRRM to remove the need to store angular boundary fluxes cannot be directly applied to deterministic MOC methods, as doing so would incur significant biases or performance costs for the following reasons:

1. In a deterministic MOC code, the tracks of the quadrature form a deterministic integral of the problem. Each track of the quadrature is responsible for a specific portion of the angular and spatial phase space. If tracks or portions of tracks are deleted from the quadrature, the angular and spatial regions those tracks were responsible for are missing from the integral, inducing bias. In the case of direct application of the methods used by TRRM (i.e., usage of a dead zone) to a deterministic MOC code, certain portions of tracks (or full tracks) will in effect be deleted from the quadrature, therefore incurring significant biases and error.
2. Theoretically, a complex re-weighting scheme might be developed to correct item (1) in order to conserve phase space. In such a scheme, an algorithm might be designed to re-assign weights of specific track segments to account for quadrature gaps caused by other nearby segments being deleted due to their conversion to dead zones. However, it is unclear if such a scheme is realistically feasible. Additionally, as correcting segment weights due to quadrature gaps caused by deletion of other segments would likely require global knowledge of the full overall quadrature, it is unlikely that such a scheme would work when using on-the-fly ray tracing, rather, the full tracking file would likely need to be pre-computed, analyzed for weight adjustments, and stored in memory.
3. The above bias issues caused by item (1) may be remedied by leveraging usage of cyclical track paths and enforcing that a full cycle is traversed after the dead zone is finished each power iteration. For example, take the case of a 100 cm cycle through a problem domain. The first 10 cm of a cycle can for instance be set to dead zone, with the remaining 90 cm of the cycle being traversed as normal in the “active” region of the ray. Then, once the cycle has been traversed once, the initial 10 cm length of track that was originally traversed as part of the dead zone can be traversed again, therefore ensuring that all tracks in the cycle were integrated exactly once.
4. The above method in item (3) for removing the bias from usage of dead zones in deterministic MOC has some limitations, however. Cyclical paths can have extremely long lengths which may make the full traversal of a cycle each power iteration impractical

computationally. For instance, if we imagine a multi-assembly scale 3D problem with a high number of quadrature angles, some angles may be very shallow, potentially requiring thousands of domain crossings before the full cycle is completed. As each crossing must be completed sequentially in order, the work to complete traversal of that cycle cannot be parallelized, resulting in an extremely long runtime for each power iteration. Additionally, as it is necessary to use domain decomposition when simulating large full core reactor problems on supercomputers[25, 26, 29, 30, 31] (as is discussed in subsection 2.5), traversal of full track cycles every power iteration may require that a ray is transmitted between subdomains handled by different processes many thousands of times each power iteration, which may incur impractical synchronization and network communication costs.

5. The above performance issues in item (4) may be remedied if the boundary conditions for a problem are limited to vacuum conditions only, as is reasonable if an application is only targeting full core nuclear reactor simulations. In this case, boundary angular fluxes do not need to be stored as all incoming angular fluxes at the boundary are zero, and no dead zones are needed as the exact angular flux is known. In this case, deterministic tracks can simply be followed once across the domain (from vacuum to vacuum) each power iteration without having to ever store any angular boundary fluxes. However, restriction of boundary conditions to vacuum only is a significant limitation and prevents usage of the application for problems featuring reflective boundary conditions such as basic pincell, assembly, and partial core problems. The ability to simulate these smaller and medium scale problems is highly desirable even for an application targeting full core simulations even if only for development, testing, and verification purposes.
6. Even if only vacuum boundary condition problems are simulated per item (5), large full core problems require domain decomposition into thousands of subdomains when running on supercomputers. In the case where all rays must start at the exterior, the majority of subdomains which lie in the interior of the problem will begin the simulation with no rays to simulate, and must wait for rays to migrate inwards. This leads to extreme computational load imbalances that would greatly inhibit parallel scalability.
7. In practice, because of the above items (1-6), actual domain decomposed deterministic MOC codes are forced to store angular boundary fluxes at each subdomain boundary between iterations[30, 31]. This results in very large boundary angular flux storage requirements, with estimates for a high fidelity simulation of the full core BEAVRS[32] benchmark ranging up to 70.5 TB when decomposed across 5,780 subdomains[30, 31]. The total amount of boundary angular flux data also increases as subdomains are divided finer, as may be necessary to scale to large supercomputers featuring tens of thousands of discrete compute nodes[33]. In contrast, the vast majority of memory used in ARRC is for storage of several copies of the scalar flux and source vectors, which are very small (likely only a few terabytes on a BEAVRS scale problem) and whose sizes are independent of the number of subdomains used in the decomposition.

The above items make clear that deterministic MOC applications can not make use of same techniques that ARRC deploys to remove the need to store angular boundary fluxes, as doing so in a deterministic application would either incur unacceptable bias penalties and/or prohibitively high performance costs. In practice, the angular flux treatments in



ARRC greatly reduce the memory requirements (typically by several orders of magnitude) as compared to traditional deterministic MOC applications. This extreme memory savings can also improve computational performance by increasing cache efficiency and reducing memory bandwidth requirements.

## 2.5. Computational Parallelism

Any method aiming to simulate entire nuclear reactors in high fidelity must be able to map efficiently onto supercomputer architectures, where many separate computers are networked together and working in tight coordination to run a simulation[34, 35, 36]. Running on a supercomputer is necessary both to reduce the time to solution and to greatly increase the total amount of memory available for the simulation in order to allow a 3D full reactor core to be simulated with a sufficiently high resolution[18, 37, 38, 39, 40, 41]. In order for a MOC or discrete ordinate (Sn) based simulation to utilize a supercomputer effectively, the domain of the reactor must be decomposed and distributed among the hundreds or thousands of computational nodes that form the supercomputer[25, 26, 29, 30, 31]. Each node is typically thought of as an independent computer featuring its own CPU chip (often with many processors), memory, and networking connection.

ARRC features three levels of parallelism: distributed memory, shared memory, and vectorization. Distributed memory parallelism is achieved by way of domain decomposition, which involves creating an efficient mapping of the geometric reactor domain onto the various computational nodes, and then creating an efficient algorithm to communicate and exchange computational data between the nodes during the simulation. The decomposition and exchange algorithms are implemented using the Message Passing Interface (MPI) library.

There are multiple viable strategies to map the reactor domain onto computational nodes (also referred to as MPI ranks). One simplistic approach is for a coarse Cartesian grid to be laid down over the reactor domain with each cubic element being assigned to a specific MPI rank to store that domain's scalar flux. For instance, each MPI rank might be responsible for a single 10 cm high assembly slice, with thousands of MPI ranks being able to collectively store an entire reactor. As rays traverse the reactor geometry during the TRRM simulation, the rays are exchanged between MPI ranks when required until each ray's terminating conditions are met. However, this can lead to a number of efficiency problems in the form of load balancing, so a more advanced approach to domain decomposition is taken in ARRC.

To understand how domain decomposition works in ARRC, it is important to make clear exactly how the reactor's geometry and spatial scalar flux distribution is stored during the simulation. In ARRC, the spatial simulation domain is defined by the constructive solid geometry (CSG) definition of the various shapes and structures within a reactor. Each CSG region also has associated multigroup cross section data. Often, large CSG regions of the reactor (for instance, a single fuel pin) are often subdivided further into smaller radial, azimuthal, and axial subregions to increase the computational resolution of the problem. The reason for this subdivision is that ARRC uses a flat source approximation to represent the scattering and fission sources inside of each CSG subregion. That is, each CSG subregion, or flat source region (FSR), has a scalar source that does not vary throughout the domain of the FSR. Any ray crossing that FSR will therefore receive a source contribution that is related only to the distance through the FSR traveled, not the ray's location or direction within the FSR. The result of this approximation is that FSRs must be very small in order to retain accuracy,

particularly in locations where the scalar flux gradient is large. The flat source approximation is commonly used in many other deterministic MOC applications[13, 18, 22, 26, 42, 43, 44], though in theory higher order sources[43, 44] could be added to ARRC at a later time to further increase the fidelity of computation.

With the FSR definition of the reactor set as input to a computation, the full domain of the problem is decomposed in ARRC by taking contiguous blocks of FSRs and assigning them to each MPI rank. The contiguous chunks are determined by way of the FSR identification numbers, which are generated to keep contiguously numbered FSRs in relatively close spatial proximity to one another. The resulting decomposition ensures that MPI domains are as continuous as possible (i.e., that all FSRs in an MPI rank are usually spatially connected in a single cluster). The advantage to this method is that any geometry can be decomposed automatically without having to overlay an additional decomposition grid which may subdivide FSRs in the problem further. Another small advantage is that it works for any number of MPI ranks and is not limited to a static decomposition or rank counts that must be perfectly distributed into a Cartesian  $P_x \times P_y \times P_z$  grid. This offers a small practical advantage in that the code can be run on an arbitrary number of computational nodes, whereas a Cartesian mesh might have to leave some nodes inactive or else resort to a sub-optimal Cartesian decomposition mesh (e.g.,  $P_x \times 1 \times 1$  where  $P_x$  is a prime number) to use all available resources. Furthermore, the distribution of FSRs to MPI ranks in ARRC can be easily altered to afford improved load balancing, which is difficult to accomplish using a strict Cartesian decomposition.

With the decomposition set, rays are sampled from a uniform random distribution in angle and space throughout the domain of the problem. As each MPI rank only contains a small portion of the global domain, a rejection sampling scheme is used wherein all MPI ranks sample the full number of rays on the global domain, but discard any rays which do not start inside their domain. This rejection sampling scheme avoids the need to have all the MPI ranks coordinate samples while ensuring that no bias is imposed on the system regardless of how the decomposition is performed. This allows for domains of dissimilar volumes and FSR densities, which may be advantageous for load balancing reasons. This also negates the need to have a strict Cartesian decomposition or bounding box for the subdomains, which is advantageous as the requirement for strict Cartesian grids can result in poor load balancing. However, the rejection sampling scheme does not explicitly enforce that all MPI ranks receive the same number of rays to start, only that the number of rays they receive corresponds statistically with the subdomain’s proportional volume out of the global domain.

Once rays have been generated and rejection sampled, the rays are then processed on their home ranks until they reach a geometric cell which resides on another processor. The rays are then queued in a buffer along with other rays that need to be sent to that MPI neighbor rank. After all ranks have processed all their rays, a bulk synchronous nearest neighbor exchange occurs. This process repeats until all rays have met their termination criteria and regular power iteration occurs. Depending on the number of MPI ranks and the size of the problem, this can require several hundred to a few thousand neighbor communication loops per power iteration.

Shared memory algorithms are also used to handle parallelism inside of each computational domain by mapping work onto the multiple CPU threads available on that node. This is achieved by using the OpenMP threading compiler language to dynamically allocate “task–



based” work (in the form of independent rays) to each thread available on the node. Each ray is composed of geometric information regarding location and direction as well as the angular flux for each neutron energy group in the simulation. Dynamic allocation of work to threads using the OpenMP dynamic scheduling option ensures load balancing on the node provided there are significantly more rays to process than there are threads. Finally, at the lowest level of parallelism ARRC uses vector instructions to ensure that the inner loop of the program (the attenuation of the angular neutron fluxes in each energy group) becomes vectorized. This potentially allows for a factor of 4-8x speedup (in terms of time per inner loop execution) depending on the Single Instruction Multiple Data (SIMD) vector width of the CPU architecture being used as well as the number of neutron energy groups used in the computation[27]. This process is summarized in Algorithm 1, which shows the major steps that ARRC performs in each power iteration of the simulation.

---

**Algorithm 1** ARRC Parallel Power Iteration Pseudocode

---

```

1: while Scalar Flux is Unconverged do                                ▷ Power Iteration Loop
2:   Sample Random Rays in Local MPI Subdomain
3:   while Rays Still Alive Globally do                                ▷ Sub-Iteration Communication Loop
4:     for all Rays in Subdomain do                                    ▷ Thread Parallelism
5:       while Ray Inside Subdomain do
6:         Ray Trace for Next Intersection
7:         for all Energy Groups do                                    ▷ SIMD Parallelism
8:           Attenuate Angular Neutron Flux
9:         end for
10:        if Ray Outside Subdomain then
11:          Buffer Ray for Exchange
12:        end if
13:        if Ray Distance > Max Distance then
14:          Terminate Ray
15:        end if
16:        if Ray Hits Global Boundary then
17:          Apply Boundary Condition                                ▷ Vacuum or Reflective
18:        end if
19:      end while
20:    end for
21:    Synchronize MPI Ranks
22:    Exchange Buffered Rays with MPI Neighbors
23:  end while
24:  MPI Reduction of Global Variables                                ▷ k, Total Ray Distances, Shannon Entropy
25: end while

```

---

The models for parallelism and communication patterns inherent to a domain decomposed MOC[25, 26, 31] or TRRM code are in many ways similar to those in domain decomposed Monte Carlo codes. For instance, the transfer of rays between adjacent computational domains inside of a power iteration in TRRM is similar to the transfer of particles between domains in a domain decomposed MC simulation. In recent years there has been significant

research into these MC communication models and costs[38, 41, 45, 46, 47, 48] that were leveraged when developing the communication model used in ARRC. However, there are some key differences in communication requirements between Monte Carlo and TRRM that can sometimes lead to domain replication being preferable in Monte Carlo instead of domain decomposition (particularly when the geometry model and all tally data are small enough to fit on a single computational node).

With the extreme memory efficiency of TRRM, domain replication is definitely possible and is available as an option for work decomposition in ARRC. However, when scaling up to larger problems, it was found that the synchronization of the scalar fluxes between all solutions every power iteration became prohibitively expensive. This synchronization is required as all processes must have the full scalar flux vector as input to the next power iteration. Processes are not capable of simply tracking their own independent scalar flux vectors and then reducing them all at the end of the simulation as the per-process ray density requirements to hit nearly all FSRs every iteration would be prohibitively expensive. Therefore, to accomplish this global synchronization usage of the MPI `MPI_Allreduce` construct every power iteration on the full scalar flux vector is required. For larger problems featuring many-GB scalar flux vectors, this began to dominate the runtime and could not compete in performance with the domain decomposed model. While the domain decomposed parallel model in ARRC does require frequent communication of rays inside of each power iteration, the communication is all done locally between neighbors which exploits data locality and leverages the high speed local connections between neighbor ranks in a supercomputer[49]. Message sizes are also very small, for instance in the 3D C5G7 benchmark problem (presented in [subsection 3.2](#)) nearest neighbors typically only exchanged 11 KB of data each communication round. Additionally, in the domain decomposed model, very little ever needs to be globally reduced between processes each power iteration, save for a few scalar values like total fission source, convergence detection metrics, and eigenvalue. Overall, similar to deterministic MOC[25, 26, 29, 30, 31], the domain decomposed model is much more efficient for TRRM on large problems.

Comparatively, Monte Carlo codes, even when domain replicated, only need to pass around minimal eigenvalue information each power iteration, with tallies only ever being required to be reduced to a single process or file once at the very end of the simulation[50]. This allows for Monte Carlo codes to scale very efficiently using domain replication. It's also worth noting that Monte Carlo codes may also eventually benefit from domain decomposition, as some estimates for high fidelity simulations require several terabytes worth of tally data[38, 41] which may be difficult to store in full on every computational node.

The domain replication parallelism model is still useful for some problems when using TRRM, particularly very small problems like pincells with few flat source regions. While such problems do not need supercomputers to solve, on-node shared memory threaded parallelism can still be desirable to speed up a simulation. Having many threads contending for shared memory locks on only a small number of flat source regions causes very poor shared memory scaling. In this situation, the domain replication model can be much faster, allowing the code to be run with MPI only instead of threads, with scalar flux vector reduction being extremely quick due to the low number of flat source regions. Domain replication is therefore also available as an option in ARRC, though results are not presented in this analysis as domain decomposition was found to be far faster for all benchmark problems presented in this paper.

## 2.6. Convergence

In each power iteration of ARRC, a new solution for the scalar flux and fission source distribution throughout the problem domain is generated. This solution will differ slightly from the previous iteration’s solution for two reasons:

1. Iterative numerical convergence towards the final, converged solution
2. Stochastic noise caused by a new set of random rays being used each iteration

Typical deterministic processes will often measure convergence by computing the root mean square (RMS) or relative RMS between successive iterations’ scalar flux vectors or fission source vectors[13]. The RMS will decrease each iteration due to numerical properties of the system being simulated. The fractional decrease in RMS at each iteration is proportional to the apparent dominance ratio of the system, which is the ratio of the second largest eigenvalue to the largest eigenvalue, with higher dominance ratio[51] problems being harder to converge and requiring more power iterations. In the limiting case of TRRM where an infinite number of random rays are used in each power iteration, there will no longer be any stochastic noise in the scalar flux or fission source each iteration and the simulation will converge according to the apparent dominance ratio in the same manner as a deterministic process. However, as only a finite number of rays can be used, there will always be stochastic noise present in the scalar flux vector generated each iteration. This means that simple comparison between of scalar flux vector solutions between two iterations using RMS difference will not converge past the level of stochastic noise. Additionally, the final scalar flux vector produced by the last power iteration, while unbiased, will be extremely noisy. Therefore, the convergence systems typically used by deterministic MOC solvers cannot be used, rather, methods of convergence used in Monte Carlo solvers are applied to TRRM in ARRC.

To begin, the simulation is split into “inactive” and “active” regions similar to Monte Carlo eigenvalue simulations[51]. The inactive region is used to satisfy the dominance ratio requirements of the problem by performing enough iterations for the scalar flux and fission source distribution to become stationary. Once the problem is judged as stationary, eigenvalue and scalar flux distributions are no longer moving and each subsequent power iteration should produce a statistically similar answer. At this point, the active region can be entered wherein statistics are accumulated by storing and averaging the scalar flux distribution and eigenvalue each power iteration. Successive iterations in the active region serve to reduce the uncertainty of all unknowns being tracked, with the standard deviation of unknowns following a  $1/\sqrt{n}$  convergence pattern, where  $n$  is the number of active iterations.

There are two difficulties introduced in this arrangement: 1) a decision must be made as to what constitutes a “stationary” source in order to begin the active phase, and 2) a convergence criteria of some sort must be selected to serve as a stopping criteria for the simulation. The easiest way of addressing these problems is to simply allow the user to select the desired number of active and inactive iterations. While this is available as an option in ARRC, it is not necessarily recommended especially when working in an unfamiliar problem space as setting the number of inactive iterations too low can result in a highly biased solution, and setting the number of active iterations too low can result in inaccurate results. Therefore, in ARRC, several methods are available to automate this process for the user and to help ensure accuracy.

To automatically determine when the simulation reaches a stationary distribution, ARRC uses a method that measures a property of the fission source distribution rather than the eigenvalue. For high dominance ratio problems, the source distribution of the system can be slower to converge than the eigenvalue[51]. This means that it is important to measure convergence of the source distribution when judging if the simulation has become stationary. While a user could in theory view a plotted picture of the source distribution of a reactor every iteration to judge when it has stopped changing, this would be extremely onerous for the user, so use of a single scalar value that represents the overall source distribution is highly preferable. Therefore, to determine when the simulation has reached a stationary source distribution, ARRC computes the “Shannon Entropy” [51, 52] (a concept from information theory) of the fission source vector after every iteration. Shannon Entropy takes the form of a single scalar value that helps to characterize the convergence of the overall source distribution. It is calculated in ARRC as:

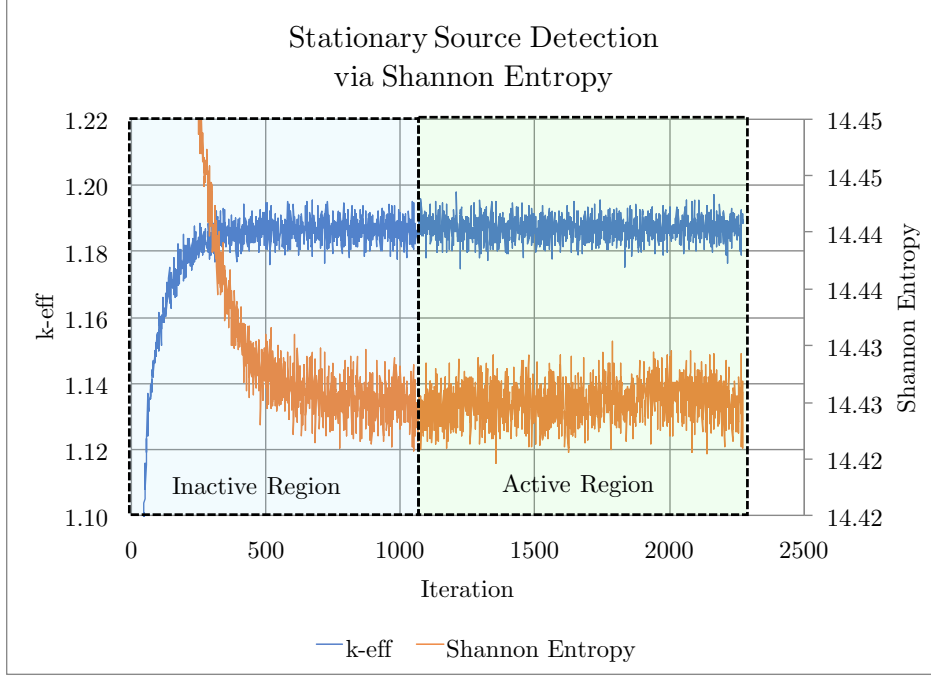
$$H_{src} = - \sum_i^N P(i) \ln_2(P(i)) \quad (1)$$

$$P(i) = \frac{V(i)}{F} \sum_g^G [\Sigma_g^f(i) \phi_g(i)] \quad (2)$$

where  $H_{src}$  is the Shannon Entropy,  $\Sigma_g^f(i)$  is the macroscopic fission cross section in spatial region  $i$  and energy group  $g$ ,  $\phi_g(i)$  is the scalar flux in region  $i$  and energy group  $g$ ,  $V(i)$  is the volume of region  $i$ ,  $F$  is the overall total fission source,  $N$  is the number of regions, and  $G$  is the number of energy groups. Using this definition, the Shannon Entropy mesh is therefore equivalent to the problem’s flat source region mesh. While Shannon entropy in itself does not measure a useful physical property of the reactor being simulated, the value does plateau at an arbitrary level when the overall source distribution becomes stationary. ARRC searches for this plateau each iteration by storing a travelling window of  $H_{src}$  over the previous iterations and testing whether the means of  $H_{src}$  in each half of the window are within one standard deviation of each other. In practice, a default window size of 200 is used in ARRC as it was found to accurately judge convergence on a wide variety of problems (including those with high dominance ratios).

A test problem (the 2D C5G7 benchmark, as discussed further in [subsection 3.1](#)) was run to demonstrate how eigenvalue and Shannon Entropy develop over the course of a simulation. As seen in [Figure 1](#), both metrics change significantly between each power iteration due to stochastic noise, but a clear trend of steep change in the early iterations followed by an eventual plateau is seen. By using Shannon Entropy, ARRC judged the source as stationary after 1058 iterations.

With the source judged as stationary and the active region begun, scalar fluxes and eigenvalues begin accumulation. Some convergence criteria must now be selected in order for the simulation to stop. While the eigenvalue could in theory be used to track convergence, as previously discussed the eigenvalue may converge more quickly than the scalar flux and source distributions, meaning that measuring convergence of the scalar flux or source distributions is necessary. In ARRC, the convergence criteria is defined in terms of the desired source average relative error (SARE). The SARE is calculated by computing the vector of all the

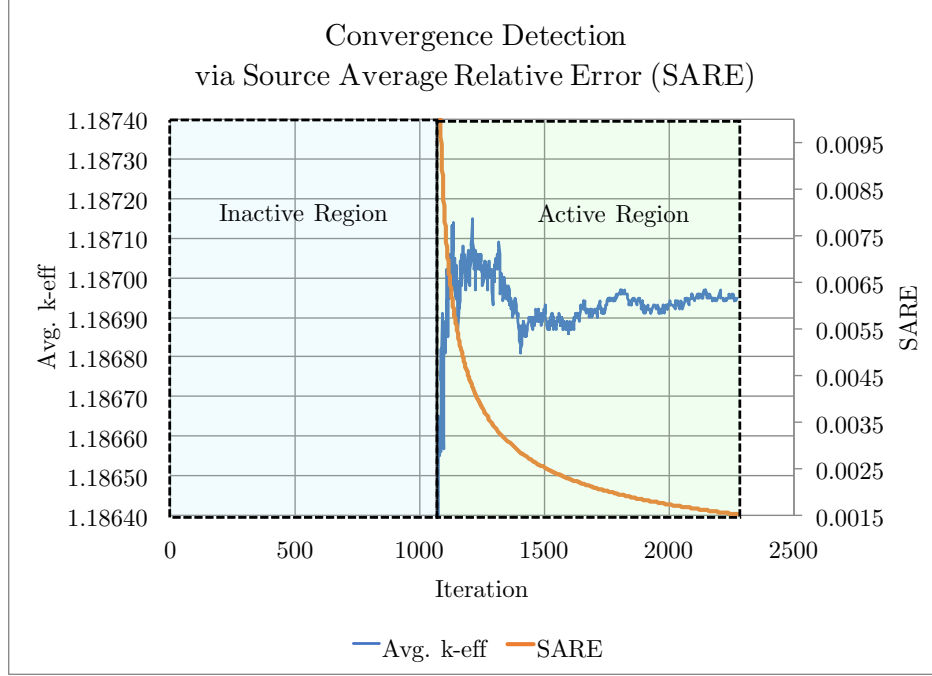


**Figure 1:** K-effective eigenvalue and Shannon Entropy vs. iteration number for an example problem. The simulation was judged as “stationary” after 1058 iterations, which was determined by monitoring the travelling window of Shannon Entropy values and checking when each half’s mean were within one standard deviation of one another.

relative standard deviations for all fission sources (one in each spatial flat source region, integrated in energy) and taking the average of this vector. It is important to note that relative rather than absolute standard deviations are used (by simply dividing the absolute standard deviation for each source by its mean) as this better ensures that the overall shape of the system is converged rather than focusing only on the higher magnitude source bins. Assuming the source distribution is perfectly stationary when beginning the active region of the computation, SARE will asymptotically follow the central limit theorem and will drop following a  $1/\sqrt{n}$  trend, where  $n$  is the number of active iterations. For example, a simulation with a desired SARE level of 0.001 will require 4 times more active iterations than the same simulation with a desired SARE level of 0.002. An example of the convergence tracking and detection process is shown in [Figure 2](#), which was generated when simulating the 2D C5G7 benchmark.

Once convergence criteria have been met, the simulation is complete and the average eigenvalue, average scalar flux vector, and average fission source vector are returned as a result to be stored to file or plotted if desired. It is important to note that during the active region, the average scalar fluxes and sources are not used in the simulation at all except for determining convergence. Each power iteration uses the actual “live” scalar flux vector produced by the previous iteration as input in a manner similar to deterministic MOC simulations.

It is also worth highlighting that management of this stochastic convergence process is an added complication compared to traditional deterministic MOC methods which feature very straightforward convergence properties. While traditional codes need only to measure the iterative RMS change between scalar flux vectors or fission source vectors to track convergence,



**Figure 2:** Average eigenvalue (k-eff) and SARE in the active region of a test problem. The SARE target for this simulation was set at 0.0015.

TRRM requires the more complex two phase convergence scheme detailed above. However, it is important to note that this stochastic neutronics convergence process is very well understood as it is similar in many ways to the convergence process seen in Monte Carlo (MC) style neutronics methods. Additionally, the convergence process of TRRM is significantly simpler when compared to MC for a number of reasons. One important simplification is the usage of multigroup cross sections which significantly reduces the variance as compared to continuous energy MC simulations. Additionally, TRRM forms an equal volume integral throughout the domain, unlike MC where all particles begin life in a fissile material, making convergence of statistics in areas remote from fuel more difficult. Finally, TRRM is not subject to any particle clustering effects that can be seen in MC, which can be caused by successive neutron generations being highly correlated in space with one another and can impede convergence of MC eigenvalue calculations[53, 54, 55]. The lack of particle clustering effects is also important as in Monte Carlo such clustering can complicate the usage of Shannon Entropy in detecting when a source has become stationary[52].

These effects combine to greatly simplify the convergence properties of TRRM as compared to MC, allowing for a relatively smooth and predictable convergence process as will be demonstrated empirically in the next section. Additionally, all parameters in ARRC regarding convergence and fidelity have associated heuristics that will be presented in [section 4](#).

### 3. Validation

In this section 2D and 3D benchmark problems are run using the new application ARRC. Accuracy results compared to reference solutions are presented in terms of eigenvalue and a variety of pin power error metrics.



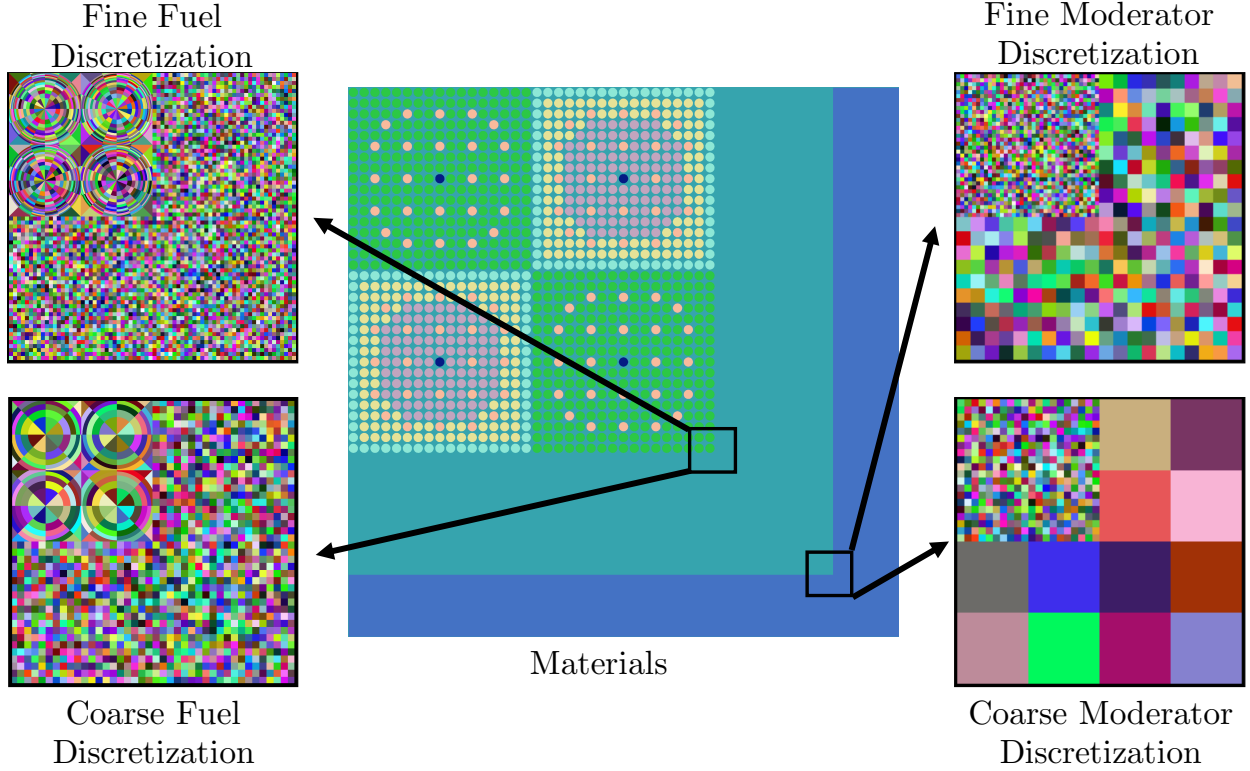
### 3.1. 2D

To validate the accuracy of the TRRM algorithm, we first test the ARRC on the 2D C5G7 reactor benchmark problem[56]. This benchmark is composed of 2 mixed oxide (MOX) and 2 uranium oxide (UOX) fuel assemblies (17x17 pins each) and 5 water reflector regions, as shown in [Figure 3](#). Note that the left and top boundaries are reflective and the bottom and right boundaries are vacuum. In this benchmark, the fuel, gap, and clad structures normally found in real world fuel pins are homogenized into a single material, so that the unit pincells in the C5G7 benchmark are composed of a single circular “homogenized fuel” pin which is surrounded by a moderator region. The benchmark features in total 5 materials with 7 group cross section data provided for each material. This benchmark is useful for several reasons. First, it features a heterogeneous geometry that is harder for codes to resolve accurately than other homogeneous benchmark problems. This geometry also features significant flux gradients due to the moderator reflector regions which can be challenging for codes to resolve accurately. Additionally, the benchmark has a reference solution that was produced via isotropic multigroup Monte Carlo using the same 7 group cross section data. This enables “apples to apples” comparisons between applications as it removes the possibility of errors resulting from multigroup cross section generation or anisotropy treatments, both of which tend to vary between applications. For these reasons, it is an excellent benchmark problem to validate the accuracy and performance of a code.

The benchmark was run in 2D using the ARRC code. Several configurations were used to better understand the relations between computational mesh and accuracy. While the benchmark clearly defines the reactor geometry, TRRM achieves much higher accuracy when the material regions are broken down into smaller cells, known as flat source regions (FSRs). This is because ARRC uses the approximation that the scalar flux (and isotropic neutron source) within each of these cells is “flat”, having no shape. In order to accurately capture the many gradients in flux throughout the pincells and moderator regions of the reactor, a finer computational mesh is used to increase the fidelity of the computation. This imposed mesh is defined using constructive solid geometry (CSG) so as to follow the contours of the geometry accurately, for instance splitting a circular fuel region into octants and several concentric regions. Two mesh configurations were used in this analysis, a “coarse” mesh and a “fine” mesh, as shown in [Figure 3](#). The coarse mesh features 142,964 FSRs while the fine mesh features 591,892 FSRs. Note that in both cases coarser FSRs were used towards the outside of the moderator region where the flux gradient is very mild compared to the steep gradients seen near the assembly-moderator interface.

In addition to running the code using multiple FSR meshes, we also varied the convergence criteria. Convergence in ARRC is measured in terms of SARE, as described in [subsection 2.6](#). Tightening the convergence criteria results in more power iterations being run before they are met, resulting in overall higher accuracy. Ideally, the gains in accuracy for each unknown should be governed by the central limit theorem which states that the variance of the unknown should fall off at a rate of  $1/\sqrt{n}$ .

Several other parameters are used to determine the computational accuracy of the code. These parameters are the number of rays per power iteration, the dead zone distance per ray, and the total distance each ray travels before termination. A discussion of optimal selection of these parameters is presented in [section 4](#).



**Figure 3:** 2D C5G7 geometry with cutouts showing samples of the two different flat source region (FSR) discretizations used in this analysis. Note that while there is only a single moderator material present in the outer region, a lower density mesh is used in the outer regions of the reflector to account for the significantly lower flux gradient than is present in the inner regions near the fuel-moderator boundary.

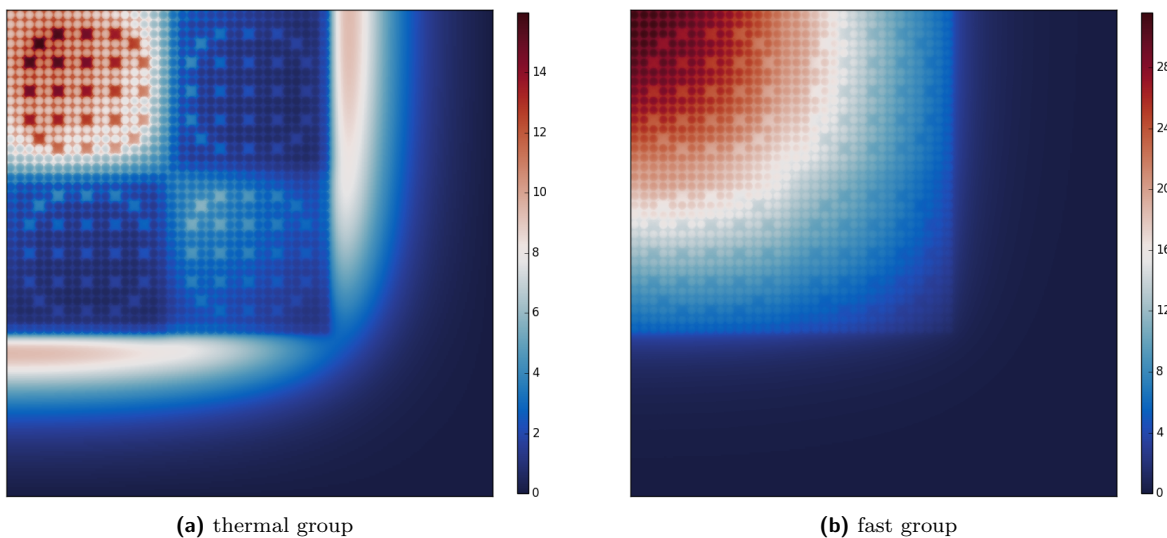
With these parameters set a number of analyses were carried out on the coarse and fine meshes while varying the convergence criteria. As ARRC is a stochastic process, the results of any single simulation will be different provided that a different random number seed is used. While the code is capable of providing an estimate of the variance and standard deviation of its solution, these estimates assume that the samples taken each power iteration are independent of those taken on previous power iterations. However, this is not necessarily an accurate assumption as each successive iteration uses the scattering and fission sources generated by the previous iteration, meaning successive iterations have the potential for correlation. The net result is the potential for the variance of unknowns to be underestimated due to correlation effects, as is seen in other stochastic methods such as Monte Carlo. Therefore, to fully understand the accuracy and variance of solutions generated by ARRC, ensemble studies are performed wherein the code is run multiple times with different random seeds. The results of each run are therefore fully independent, so the true variances can be accurately determined. In this study, 100 ensemble runs were performed for each configuration. Further testing to determine the quality of variances reported by single ARRC simulations is performed in [subsection 3.3](#), where it is shown that in practice ARRC does not underestimate the true variances.

With these parameters set, ensemble studies were performed for various configurations with the results tabulated in [Table 1](#). All tests were performed on a dual socket Intel

Broadwell-EP Xeon E5-2699 v4 node with 44 CPUs total. A number of error metrics are given for the results, as compared to the reference solution. The eigenvalue (k-effective) is a global metric for the problem which indicates the growth of the neutron population between subsequent neutron generations, with a reference solution value of 1.18655. The remaining error metrics describe the error in the pin power distribution for the 1056 fueled pins in the problem. Average percent error (APE) is the average (absolute) percent pin power error for the fueled pins. Mean relative error (MRE) is the average (absolute) percent pin power error but scaled by the reference power level of each pin, therefore weighting the higher power pins more heavily. The root mean square error (RMS) is the standard root mean square of the pin power percent error distribution. The maximum absolute percent error (MPE) is the maximum percent error value in pin power from any of the 1056 fueled pins. The hot and cold pin power errors are defined by the percent error between the hottest or coldest pin in the reference solution and the pins in those same locations in the ARRC solution. The corner pin power ratio is defined as the ratio of the innermost fuel pin (top left) to outermost fuel pin (bottom right) in the problem, and is useful for determining how well the code has captured the flux gradient from the middle of the reactor core to its edge. The standard deviation is given for all these metrics as computed using the 100 independent ensemble runs.

A figure of merit is also given in the form of the total number of *integrations* performed in the simulation across all power iterations. This metric is defined as the total number of times that an energy group of angular flux is attenuated through a geometric flat source region (FSR) of the domain. The number of integrations is equivalent to the number of evaluations of line 8 of Algorithm 1, which forms the innermost loop of the program. The total number of integrations is a useful metric for algorithmic efficiency comparison to other TRRM or MOC style applications as unlike runtime comparisons the metric removes any programming optimization discrepancies between applications. I.e., the total number of integrations to converge a solution is a function of algorithmic performance alone.

Using the fine FSR mesh the scalar flux solution is displayed using ARRC's plotting capabilities in Figure 4 for the fast and thermal groups.



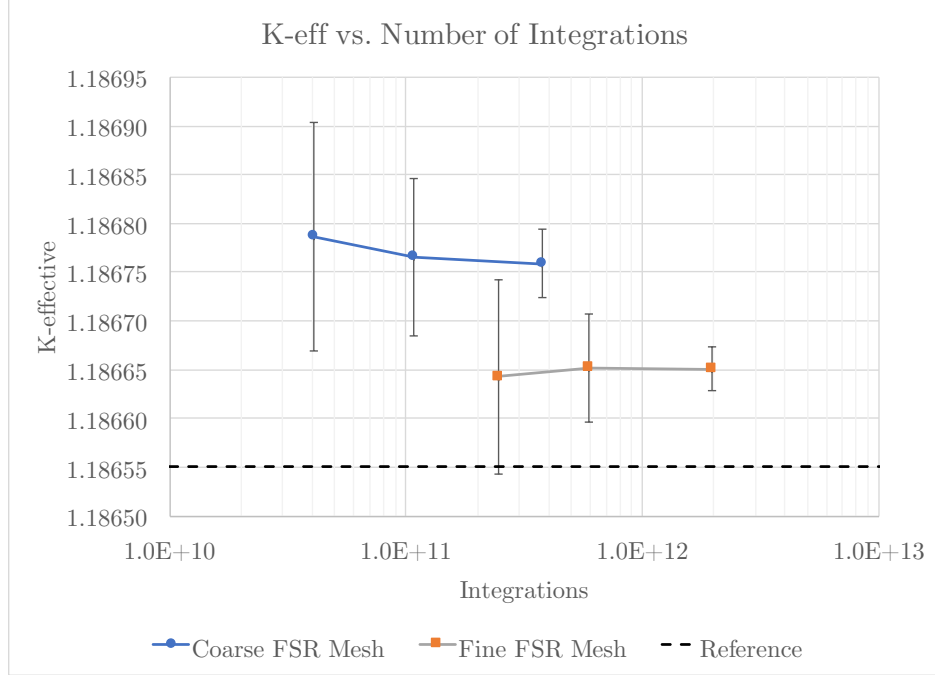
**Figure 4:** 2D C5G7 flux distribution using the “fine” FSR mesh and 0.00075 SARE convergence criteria.

	Coarse Flat Source Region Mesh			Fine Flat Source Region Mesh		
FSRs	142,964			591,892		
Memory Usage [MB]	19.9			82.4		
Number of Rays per Iteration	650			2000		
Dead Zone Distance [Mean Free Paths]	2			2		
Dead Zone Distance [cm]	12.56			12.56		
Distance per Ray [Mean Free Paths]	100			100		
Distance per Ray [cm]	628.12			628.12		
Independent Ensemble Runs	100			100		
Convergence Criteria [SARE]	0.00300	0.00150	0.00075	0.00300	0.00150	0.00075
k-eff	1.186787	1.186766	1.186759	1.186643	1.186652	1.186651
k-eff std. dev.	0.000117	0.000081	0.000036	0.000100	0.000056	0.000023
k-eff Error vs. Reference [pcm]	19.97	18.20	17.61	7.84	8.60	8.51
Average Percent Error (APE) [%]	0.353479	0.260546	0.223400	0.229901	0.174422	0.149393
APE std. dev. [%]	0.046185	0.016313	0.008031	0.024338	0.011968	0.003977
Mean Relative Error (MRE)	0.319041	0.231342	0.194080	0.207119	0.155687	0.130486
MRE std. dev.	0.036150	0.013156	0.007743	0.019009	0.010062	0.003503
Root Mean Square Error (RMS)	0.450181	0.335743	0.291117	0.291316	0.221518	0.190427
RMS std. dev.	0.058433	0.021661	0.010479	0.030547	0.014982	0.004645
Maximum Absolute Percent Error (MPE) [%]	1.689696	1.348680	1.174692	1.067192	0.817096	0.692932
MPE std. dev. [%]	0.249940	0.147981	0.095966	0.145168	0.110072	0.043406
Hot Pin Power	2.492887	2.493558	2.492986	2.496654	2.496435	2.496804
Hot Pin Power std. dev.	0.006086	0.004355	0.001838	0.004611	0.002475	0.001557
Hot Pin Power Error [%]	-0.192583	-0.165720	-0.188607	-0.041774	-0.050517	-0.035741
Hot Pin Power Error std. dev. [%]	0.243669	0.174356	0.073602	0.184607	0.099077	0.062332
Cold Pin Power	0.233765	0.233811	0.233852	0.232619	0.232598	0.232558
Cold Pin Power Std. dev.	0.001010	0.000636	0.000279	0.000723	0.000386	0.000157
Cold Pin Power Error [%]	0.977230	0.997111	1.014751	0.481887	0.473144	0.455719
Cold Pin Power Error std. dev. [%]	0.436412	0.274710	0.120577	0.312168	0.166739	0.067720
Corner Pin Power Ratio	7.599876	7.606127	7.598495	7.643598	7.642955	7.643009
Corner Pin Power Ratio std. dev.	0.040505	0.025450	0.012464	0.026247	0.016757	0.008103
Corner Pin Power Ratio Error [%]	-0.993118	-0.911673	-1.011098	-0.423530	-0.431904	-0.431201
Corner Pin Power Ratio std. dev. [%]	0.527681	0.331543	0.162376	0.341929	0.218295	0.105566
Avg. Total Power Iterations	2318	6167	21561	2009	4825	15916
Avg. Inactive Power Iterations	1038	1038	1045	1082	1110	1054
Avg. Active Power Iterations	1280	5128	20516	927	3715	14862
Avg. Number of Integrations	4.05E+10	1.08E+11	3.77E+11	2.47E+11	5.94E+11	1.96E+12
Avg. Runtime [s]	60.87	157.88	552.54	274.53	656.24	2166.36
Avg. Runtime [Intel core-hours]	0.74	1.93	6.75	3.36	8.02	26.48

**Table 1:** 2D C5G7 results summary.

A comparison of eigenvalue convergence is given in [Figure 5](#) for both coarse and fine FSR meshes with varying convergence criteria. This figure shows that for a given FSR mesh, the convergence criteria does not further bias the solution. Rather, increasing the convergence criteria simply reduces the variance in the solution. This can be contrasted against existing deterministic methods wherein the solution travels, so loosening the convergence criteria often means biasing the solution in the direction that the problem is converging from. [Figure 5](#) also shows that k-effective approaches the reference solution as a function of the fineness of the FSR mesh. This is expected, as the flat source approximation has difficulty accurately capturing the extremely steep gradients seen in the problem, so further refinement in the FSR mesh should progressively approach the reference solution value. [Figure 6](#) shows the standard deviation data for the coarse FSR mesh against the ideal  $1/\sqrt{n}$  convergence predicted

by the central limit theorem, which compares very well and strongly indicates that the TRRM method does indeed follow the central limit theorem. This is in contrast to other stochastic methods, such as Monte Carlo, wherein particle clustering and fission bank autocorrelation can inhibit  $1/\sqrt{n}$  scaling[53, 54, 55].

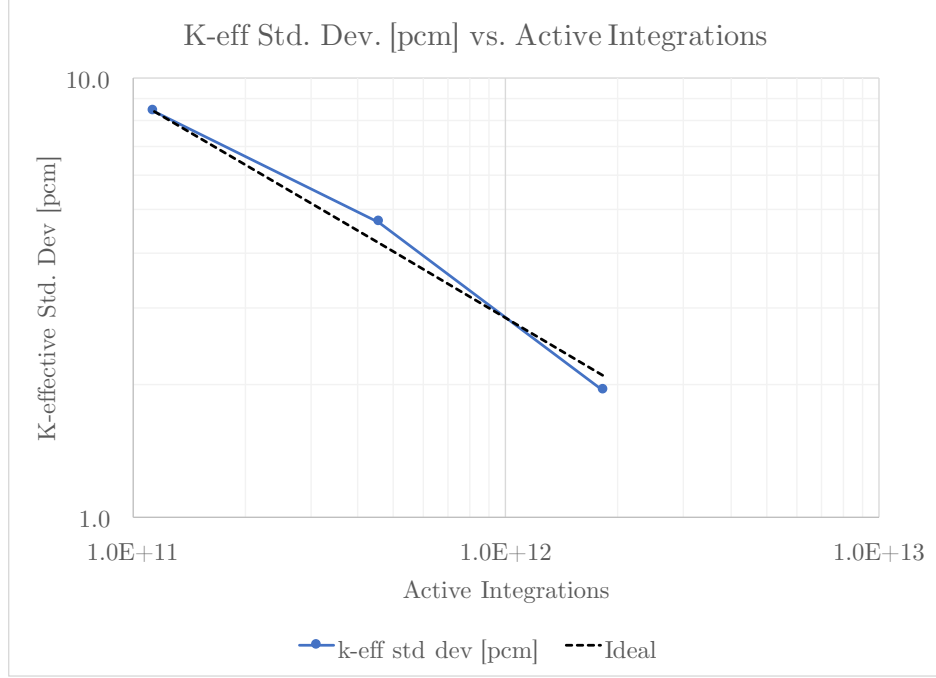


**Figure 5:** K-effective vs. total number of integrations in the simulation for 2D C5G7. K-eff data points are given as the mean of 100 independent ensemble simulations of the problem with that configuration, with error bars showing the standard deviation of k-eff in the ensemble. Data is shown for both the coarse and fine FSR discretizations with varying convergence criteria.

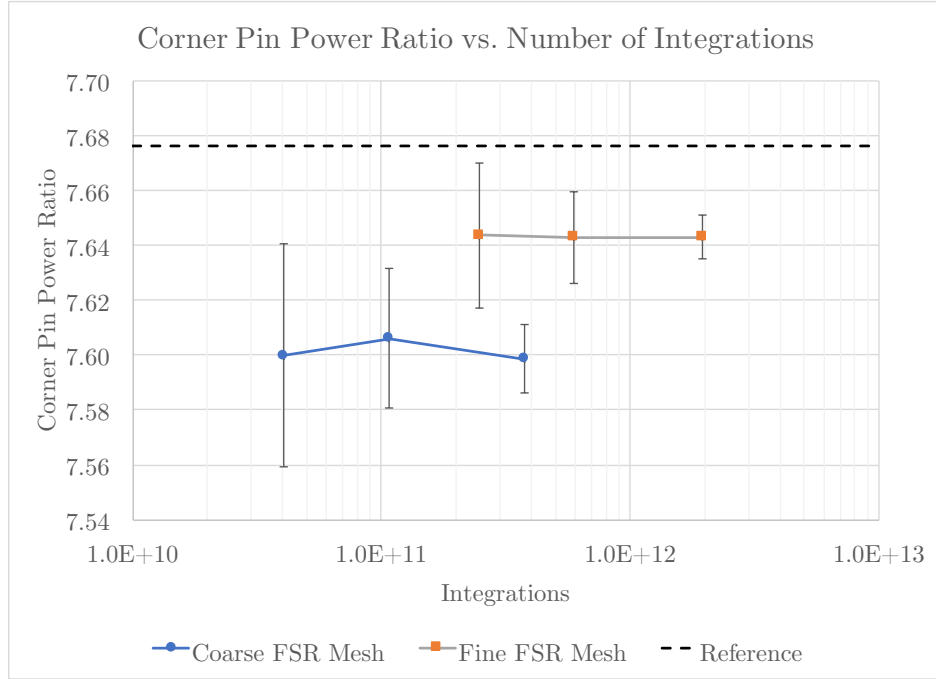
An illustrative metric of the pin power distribution, the corner pin power ratio, is displayed in Figure 7 for the various problem configurations. Similarly to the eigenvalue, the pin power ratio converges with TRRM to the asymptotic mean, though results are always normally distributed around this mean for a given FSR mesh. Also similarly to eigenvalue, the standard deviation of the corner pin power ratio drops off with  $1/\sqrt{n}$  as shown in Figure 8.

Finally, the overall pin power error distribution for the fueled region of the 2D C5G7 benchmark problem is given in Figure 9 for the coarse and fine FSR meshes and 0.00075 SARE convergence criteria. Note that the white gaps are the guide tubes in the problem, which do not contain fuel and therefore do not generate power and are not counted in any of the power distribution metrics. As can be seen, the largest errors generally occur at the assembly-moderator interfaces, with the largest percent error in the distribution being 1.17% for the coarse FSR mesh and 0.69% for the fine FSR mesh.

Using the results presented in this section, a general assessment of 2D performance and accuracy can be made by comparing to an existing traditional deterministic 2D MOC code, OpenMOC. Published results[13] from OpenMOC for the 2D C5G7 benchmark problem are compared against ARRC in Table 2. Both regular power iteration as well as Coarse Mesh Finite Difference (CMFD)[57] accelerated iteration results from OpenMOC are presented. As can be seen, ARRC is significantly faster than unaccelerated OpenMOC (in terms of core-hours) while simultaneously producing more accurate results in terms of average relative

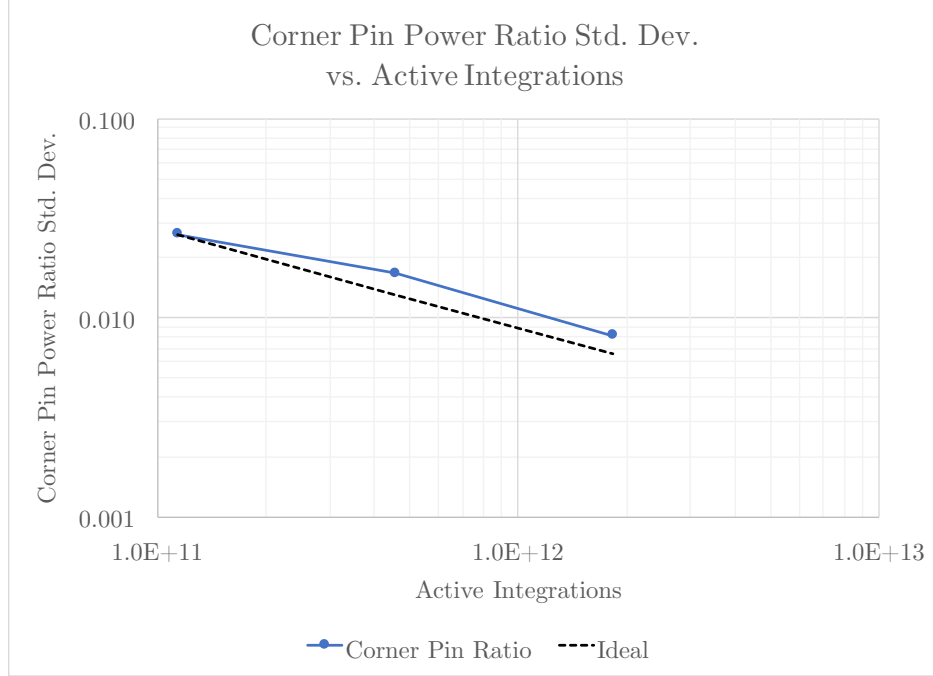


**Figure 6:** Standard deviation of k-eff vs. ideal  $1/\sqrt{n}$  behavior, where  $n$  is the number active of integrations. Results were generated using the coarse FSR discretization varying the convergence criteria. Each data point is the result of 100 independent ensemble simulations.

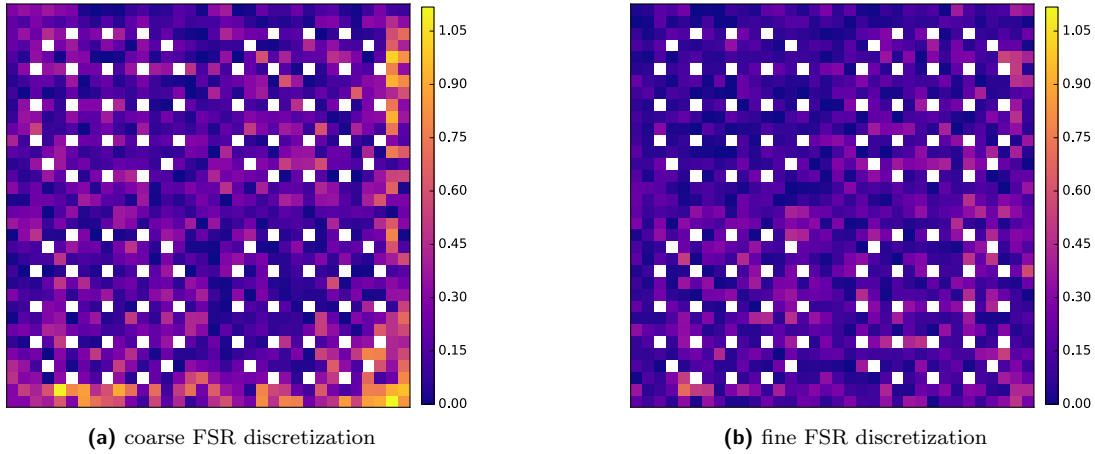


**Figure 7:** The corner pin power ratio vs. total number of integrations in the simulation for 2D C5G7. Corner pin power data points are given as the mean of 100 independent ensemble simulations of the problem with that configuration, with error bars showing the standard deviation of the pin power in the ensemble. Data is shown for both the coarse and fine FSR discretizations with varying convergence criteria.





**Figure 8:** Standard deviation of the corner pin power ratio vs. ideal  $1/\sqrt{n}$  behavior, where  $n$  is the number of active integrations. Results were generated using the coarse FSR discretization varying the convergence criteria. Each data point is the result of 100 independent ensemble simulations.



**Figure 9:** 2D C5G7 pin power error [%] distribution for coarse and fine FSR meshes with 0.00075 SARE convergence criteria. Error is computed as compared to the benchmark reference solution. These plots only show the fueled region of 2D C5G7 as seen in Figure 3.

pin power error (APE) and maximum pin power relative error (MPE). With CMFD enabled in OpenMOC, runtimes were reduced by a factor of about 22x. Compared to accelerated OpenMOC, the coarsest configuration of ARRC was only 7% slower but produced a more accurate solution in terms of APE and MPE. Overall, these results show that on the 2D C5G7 benchmark problem, TRRM offers good speed and accuracy as compared to CMFD accelerated deterministic 2D MOC.

It is also possible that ARRC may benefit from CMFD or similar acceleration methods, but such methods have not yet been implemented in ARRC. Additionally, it is important

to highlight that C5G7 is a low energy group count benchmark problem, having only seven energy groups. When moving to higher fidelity simulations featuring on the order of 100 energy groups[28] the ray tracing costs in ARRC will be further amortized likely resulting in improved performance, as discussed in subsection 5.1.

	Azimuthal Angles	Runtime [s]	Runtime [core-hr]	k-eff	k-eff error [pcm]	APE	MPE
ARRC	N/A	61	0.74	1.18679 +/- 0.00012	20	0.353 +/- 0.046	1.690 +/- 0.250
	N/A	158	1.93	1.18677 +/- 0.00008	18	0.261 +/- 0.016	1.349 +/- 0.148
	N/A	553	6.75	1.18676 +/- 0.00004	18	0.223 +/- 0.008	1.175 +/- 0.096
OpenMOC	4	508	1.69	1.18543	-112	1.791	6.558
	8	788	2.63	1.18456	-199	0.404	1.659
	16	1452	4.84	1.18499	-151	0.368	1.720
	32	2515	8.38	1.18625	-25	0.436	1.734
	64	4655	15.52	1.18650	-5	0.451	1.772
	128	8884	29.61	1.18663	8	0.456	1.785
OpenMOC w/CMFD	64	207	0.69	1.18659	3	0.451*	1.772*

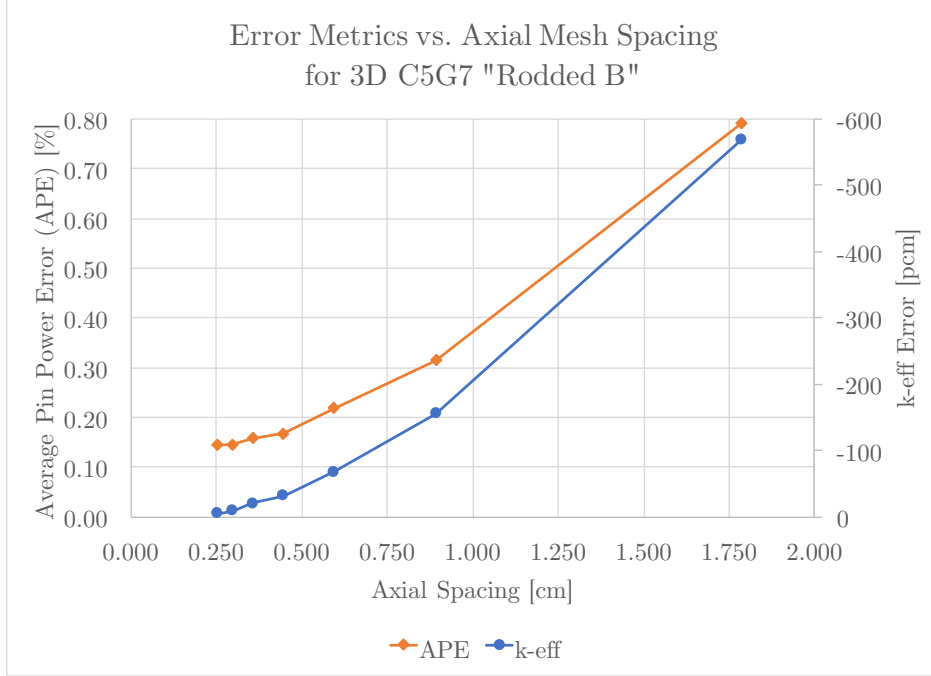
**Table 2:** 2D C5G7 performance and accuracy comparison to published results[13] from the deterministic MOC code OpenMOC. Both codes used a similar mesh featuring 142,964 FSRs. The uncertainties for results from ARRC indicate the standard deviation as measured by independent ensemble simulations. (\*) APE and MPE values for the CMFD configuration were not given, but are assumed to be the same as run without acceleration.

### 3.2. 3D C5G7

In this section, we test the ARRC code on the 3D C5G7 “Rodded B” benchmark[58]. This benchmark is similar to the 2D C5G7 benchmark pin and moderator configuration, but features an axial length of 64.26 cm. The benchmark also features a moderator region on the top of the fuel and a number of control rods inserted at different lengths into the various assemblies of the problem. The addition of the control rods to the problem results in a number of sharp axial gradients to the problem that pose an increased challenge for applications to resolve accurately. We selected the “Rodded B” version of the benchmark, as opposed to the “Unrodded” or “Rodded A” configurations, because the “Rodded B” version features the greatest amount of axial heterogeneity and is therefore the hardest to resolve accurately.

As discussed in the previous section, the fineness of the flat source region (FSR) mesh used has a significant effect on the accuracy of the computation. While the relationship between radial mesh density and accuracy in the radial direction is well documented by the results of the previous section, it was important to understand the relationship between axial mesh density and accuracy for the 3D problem. To this end, a variety of axial mesh densities were tested as shown in Figure 10. Here we find that eigenvalue error continues to drop down until 216 axial cuts are used (i.e., an axial zone spacing of 0.2975 cm). However, the pin power metrics appear to plateau when 180 axial cuts are used.

With this information, we defined two test problems to run: a “coarse” mesh and a “fine mesh”. The coarse mesh looks similar radially to the 2D coarse mesh used in the previous section, but with 180 axial cuts resulting in a total of 25,216,740 FSRs. Note that the number of FSRs is not equivalent to the number of 2D FSRs in the coarse mesh times 180, as there are a different number of radial FSRs used in the moderator region above the assemblies in



**Figure 10:** 3D C5G7 error metrics vs. axial spacing of FSRs. The radial FSR distribution in the fueled axial region of the reactor is held constant and is the same as the “fine” configuration used in the 2D studies (shown in Figure 3).

the 3D problem. For the fine mesh, a similar radial mesh is used to the 2D fine mesh from the previous section with 216 axial cuts resulting in a total of 119,205,048 FSRs.

Additionally, the number of rays used for each configuration was increased from the 2D problem by an approximately equivalent factor to the increase in the number of total FSRs. As the 3D problem is significantly more computationally expensive, only a single convergence criteria was used for each configuration. Only 10 ensemble runs were used for each configuration due to a limited allocation on the supercomputer used for the analysis. With these parameters set, the ensemble studies were performed for various configurations with accuracy and error metrics as compared to the reference solution tabulated in Table 3 and runtime performance results tabulated in Table 4. The eigenvalue of the benchmark reference solution is 1.07777.

While the ensemble runs were all performed in parallel on the IBM Blue Gene/Q supercomputer *Mira*[33] to generate the accuracy data found in Table 3, performance tests were also run on a variety of single node shared memory HPC architectures to determine the performance of ARRC on more commonly used architectures besides Blue Gene/Q, as shown in Table 4. As the average total number of integrations to converge these problems is known from Table 3, the full problems were not fully converged on each architecture. Rather, they were each run for 50 power iterations using the same configurations as detailed in Table 3 and the time per integration was calculated as the total runtime (including iteration costs etc) divided by the number of integrations performed. With the times per integration known for each architecture and problem, the projected average runtime and core-hour costs for a fully converged solution could then be computed with high accuracy. There is no measurable difference in runtime between the active and inactive iterations in ARRC. Overall, ARRC ran much more efficiently (in terms of core-hours) on the Intel platforms, for instance using

	Coarse Flat Source Region Mesh	Fine Flat Source Region Mesh
2D Midplane FSRs	142,964	591,892
Axial Cuts	180	216
Total 3D FSRs	25,216,740	119,205,048
Memory Usage [MB]	3,511	16,598
Number of Rays per Iteration	117,000	276,500
Dead Zone Distance [Mean Free Paths]	2	2
Dead Zone Distance [cm]	12.56	12.56
Distance per Ray [Mean Free Paths]	100	100
Distance per Ray [cm]	628.12	628.12
Blue Gene/Q Nodes	256	512
Independent Ensemble Runs	10	10
Convergence Criteria [SARE]	0.003	0.004
k-eff	1.077645	1.077678
k-eff std. dev.	0.000014	0.000008
k-eff Error vs. Reference [pcm]	-11.60	-8.54
Average Percent Error (APE) [%]	0.275893	0.145928
APE std. dev. [%]	0.008900	0.006598
Mean Relative Error (MRE)	0.254273	0.136250
MRE std. dev.	0.008852	0.006621
Root Mean Square Error (RMS)	0.352315	0.189591
RMS std. dev.	0.010507	0.007838
Maximum Absolute Percent Error (MPE) [%]	1.238287	0.752222
MPE std. dev. [%]	0.031266	0.032965
Hot Pin Power	1.758916	1.762228
Hot Pin Power Std. dev.	0.000813	0.000497
Hot Pin Power Error [%]	-0.384186	-0.196658
Hot Pin Power Error Std. dev. [%]	0.046043	0.028170
Cold Pin Power	0.347326	0.345431
Cold Pin Power Std. dev.	0.000131	0.000129
Cold Pin Power Error [%]	0.937525	0.386785
Cold Pin Power Error Std. dev. [%]	0.038176	0.037521
Corner Pin Power Ratio	4.262283	4.288402
Corner Pin Power Ratio Std. dev.	0.004042	0.002621
Corner Pin Power Ratio Error [%]	-1.248298	-0.643147
Corner Pin Power Ratio Std. dev. [%]	0.093649	0.060731
Avg. Total Power Iterations	2519.60	2102.50
Avg. Inactive Power Iterations	1525.10	1395.50
Avg. Active Power Iterations	994.50	707.00
Avg. Number of Integrations	9.13E+12	3.66E+13

**Table 3:** 3D C5G7 accuracy results summary.

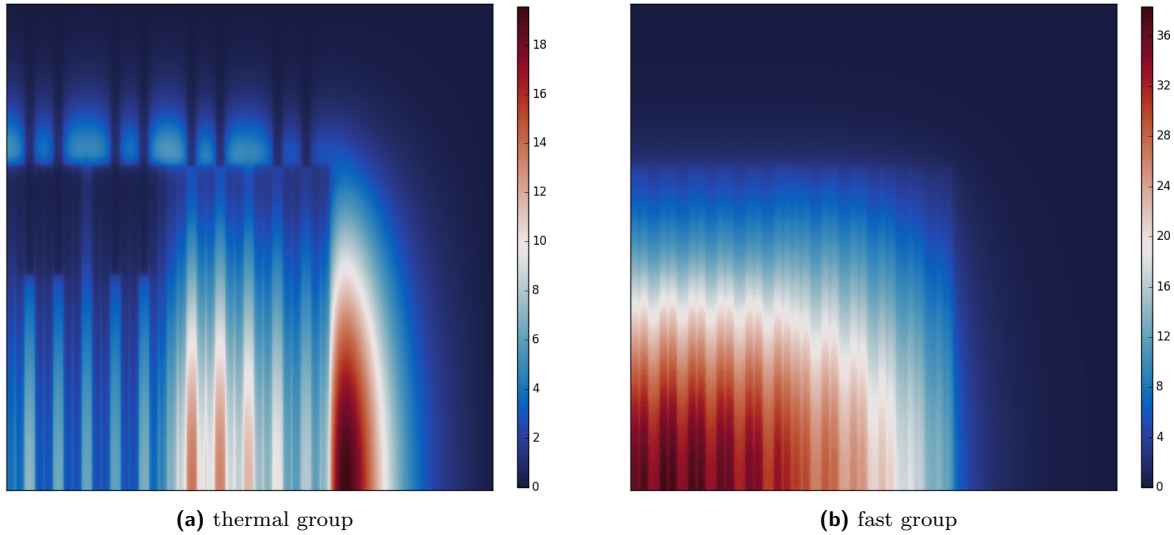
7.6x fewer core-hours on the dual socket Intel Xeon Platinum 8176 Skylake node as compared to the single IBM Blue Gene/Q node.

Using the fine mesh, an axial profile of the scalar flux solution is displayed using ARRC’s plotting capabilities in [Figure 11](#) for the fast and thermal groups. This view is the “Section A-A” slice shown in the benchmark specification[58], which cuts through two assemblies with control rods inserted to different lengths.

Finally, the overall axially integrated pin power error distribution for the fueled region of the 3D C5G7 benchmark problem is given in [Figure 12](#) for the coarse and fine FSR meshes. Note that the white gaps are the guide tubes and control rods in the problem,

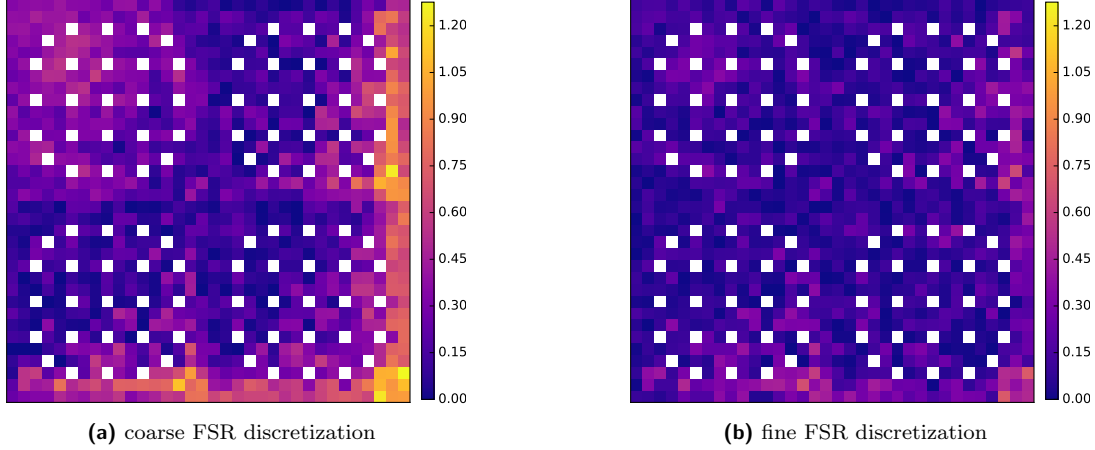
		Coarse Flat Source Region Mesh	Fine Flat Source Region Mesh
Avg. Number of Total Integrations		9.13E+12	3.66E+13
Distributed Memory Domain Decomposition + Shared Memory			
IBM Blue Gene/Q	Nodes	256	512
	CPUs	4096	8192
	Avg. Time per Integration [ns]	0.161	0.085
	Avg. Runtime [s]	1471	3100
	Avg. Runtime [Blue Gene/Q core-hours]	1674	7055
Shared Memory Only			
IBM Blue Gene/Q	CPUs	16	16
	Avg. Time per Integration [ns]	23.863	24.531
	Avg. Runtime [s]	217967	897578
	Avg. Runtime [Blue Gene/Q core-hours]	969	3989
Intel Xeon Phi 7250 (Knights Landing)	CPUs	68	68
	Avg. Time per Integration [ns]	1.922	2.019
	Avg. Runtime [s]	17553	73876
	Avg. Runtime [Intel Phi core-hours]	332	1395
2x Intel Xeon E5-2699 v4 (Broadwell-EP)	CPUs	44	44
	Avg. Time per Integration [ns]	1.350	1.451
	Avg. Runtime [s]	12332	53089
	Avg. Runtime [Intel Broadwell-EP core-hours]	151	649
2x Intel Xeon Platinum 8176 (Skylake)	CPUs	56	56
	Avg. Time per Integration [ns]	0.891	0.948
	Avg. Runtime [s]	8142	34690
	Avg. Runtime [Intel Skylake core-hours]	127	540

**Table 4:** 3D C5G7 performance results on a variety of different computer architectures. The figure of merit for this comparison, the average number of core hours required to converge the problem, is highlighted in light green.



**Figure 11:** 3D C5G7 axial flux distribution for “Section A-A” using the fine FSR mesh and 0.004 SARE convergence criteria.

which do not contain fuel and therefore do not generate power and are not counted in any of the power distribution metrics. As can be seen, the largest errors generally occur at the assembly-moderator interfaces, though some increase in error is also noticed in the inner most (upper left) assembly. This is due to the sharp axial gradients seen directly below the tip of the control rods present in that assembly.



**Figure 12:** 3D C5G7 fuel pin power error [%] distribution for coarse and fine FSR meshes. The coarse mesh was converged to 0.003 SARE and the fine mesh was converged to 0.004 SARE.

Using the results presented in this section, a general assessment of performance and accuracy can be made by comparing to an existing traditional deterministic 3D MOC code, MPACT[14]. Published results[25] from MPACT are compared against ARRC in Table 5. As can be seen, ARRC is faster than MPACT (in terms of core-hours) while also producing a more accurate solution in terms of k-effective and power distribution metrics such as APE, MRE, and MPE. Overall, these results show that on the 3D C5G7 “Rodded-B” benchmark problem, TRRM offers good speed and accuracy as compared to deterministic 3D MOC.

Code	Platform	Runtime [core-hr]	FSR Mesh	k-eff	k-eff error [pcm]	APE	MRE	MPE
ARRC	IBM Blue Gene/Q	1,674	Coarse	1.07765 +/- 0.00001	-12	0.276 +/- 0.009	0.254 +/- 0.009	1.238 +/- 0.031
	Intel Xeon (Skylake)	127						
	IBM Blue Gene/Q	7,055	Fine	1.07768 +/- 0.00001	-9	0.146 +/- 0.007	0.136 +/- 0.007	0.752 +/- 0.033
	Intel Xeon (Skylake)	540						
MPACT 3D-MOC	Titan (AMD Opteron)	10,086	-	1.07624	-153	0.325	0.267	1.182

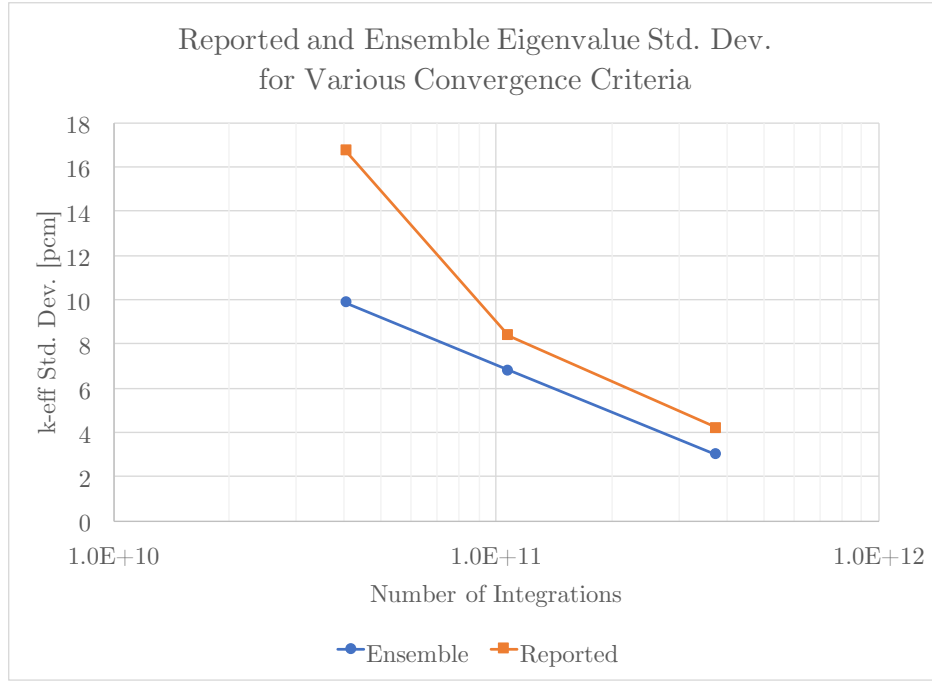
**Table 5:** 3D C5G7 “Rodded-B” performance and accuracy comparison to published results[25] from the deterministic 3D MOC module of the MPACT code, which was run on the Cray XK7 supercomputer *Titan*[59]. Runtime results for ARRC are given both for a domain decomposed parallel run on the IBM Blue Gene/Q supercomputer *Mira*[33] as well as a single node run on a dual socket Intel Xeon Platinum 8176 (Skylake) node. The uncertainties for results from ARRC indicate the standard deviation as measured by independent ensemble simulations.

### 3.3. Statistical Correlation

All studies of 2D and 3D C5G7 in this section have included a measure of the standard deviation of unknowns reported by ARRC, gathered by way of ensemble statistics generated from multiple independent simulations. While ARRC is able to directly report the standard deviation of an unknown, ensemble statistics were used to guarantee the accuracy of the standard deviations, as the iterative process used in ARRC has the theoretical potential for correlation between iterations. Therefore, a study was performed wherein the standard deviation of the eigenvalue provided by ARRC for a single simulation is compared to the true ensemble standard deviation. This study was performed on the 2D C5G7 problem using a variety of convergence criteria. As can be seen from the results shown in Figure 13, ARRC actually slightly overpredicts the standard deviation rather than underpredicting it.



This suggests that any correlation effects caused by successive iterations using the previous iteration's source distribution are trivial when compared to the randomness caused by the new ray selection. The small overprediction in standard deviation is likely caused by the source distribution not being perfectly stationary when the active region begins, causing the mean of the distribution to move very slightly (perhaps only one or two pcm) over the course of the many iterations performed in the active region, which will cause a corresponding slight increase in reported standard deviation.



**Figure 13:** A comparison of the standard deviation of the eigenvalue for a single simulation as reported by ARRC as compared to the true standard deviation generated using 100 ensemble runs. These statistics were generated on the 2D C5G7 coarse mesh problem using the same three different convergence criteria used to generate Table 1. The values are plotted in terms of the total number of integrations that were required for convergence to each criteria (i.e., tighter criteria required more integrations to converge).

The key takeaway from this study is that the standard deviations of unknowns reported by ARRC are reasonably accurate and ensemble studies are therefore unnecessary for these particular problems. This is an important point, as in other stochastic neutronics methods, such as Monte Carlo, the variance statistics reported by a single simulation can be very inaccurate due to correlation effects causing significant underestimation of the true variances[53, 54, 55]. These correlations are generally caused by particle clustering effects, usage of fission site banks, and the concept of successive particle generations that are all absent in TRRM and ARRC. However, it is important to note that the accuracy of variance reporting in ARRC still needs to be further validated using ensemble studies when full core problems are first run. This need for additional testing is due to the much higher dominance ratios seen in full core light water reactor problems which will alter the convergence properties of the simulation and could necessitate further development of more advanced stationary source detection heuristics.

The accurate variance statistics reported by ARRC are also advantageous even when compared to traditional deterministic MOC methods, as deterministic methods offer no

estimation as to the error of reported unknowns. Selection of various quadrature related parameters, such as ray spacing and the number of azimuthal and polar angles to use, can impact the accuracy of a traditional deterministic simulation significantly. Therefore, refinement studies must always be performed on these parameters when running a simulation in an unfamiliar problem space to assess each parameter’s impact on overall error. While standard quadrature parameters are often usable in a variety of problem spaces, improper usage of these default parameters may result in errors of over 2000 pcm on eigenvalue[1, 60] when used on problems featuring modern fuel designs such as integral fuel burnable absorbers. Additionally, convergence studies must be performed on traditional deterministic MOC simulations wherein the convergence criteria are altered to determine the loosest acceptable criteria. The net effect is that the general accuracy characteristics of a traditional deterministic MOC simulation in an unfamiliar problem space cannot be reliably estimated until robust parameter sensitivity analyses and convergence studies have been performed. In contrast, any convergence criteria or ray count selection in ARRC will result in both a relatively unbiased solution as well as a reliable assessment as to the accuracy of that solution in terms of its standard deviation. This limits the necessity to run convergence and error studies on a wide variety of quadrature parameters, as instead ARRC allows the user to simply tighten the convergence criteria until the standard deviations of any important unknowns become acceptable to them. On the other hand, ARRC can be subject to bias if poor choices of ray length or dead zone distances are made by the user, though as these metrics can be automatically determined in ARRC by the cross sections used in the simulation (as discussed in the next section) this issue is significantly mitigated. Also, similar to traditional deterministic MOC methods, ARRC is still subject to bias induced from the user’s choice of FSR discretization and multigroup neutron cross section data.

#### 4. Selection of Optimal Parameters

In any simulation, a wide variety of parameters must be set in order to specify the desired computational resolution. For instance, in deterministic MOC codes this is often defined by selecting the track spacing, number of azimuthal angles, number of polar angles, and the convergence criteria[13, 22, 26, 43]. In Monte Carlo (MC) codes, parameters often include the number of active batches, the number of inactive batches, and the number of neutrons per batch[9]. In ARRC, the key parameters are:

- the convergence criteria
- the distance each ray should travel before termination
- the dead zone distance for the ray
- the number of random rays to run in each power iteration

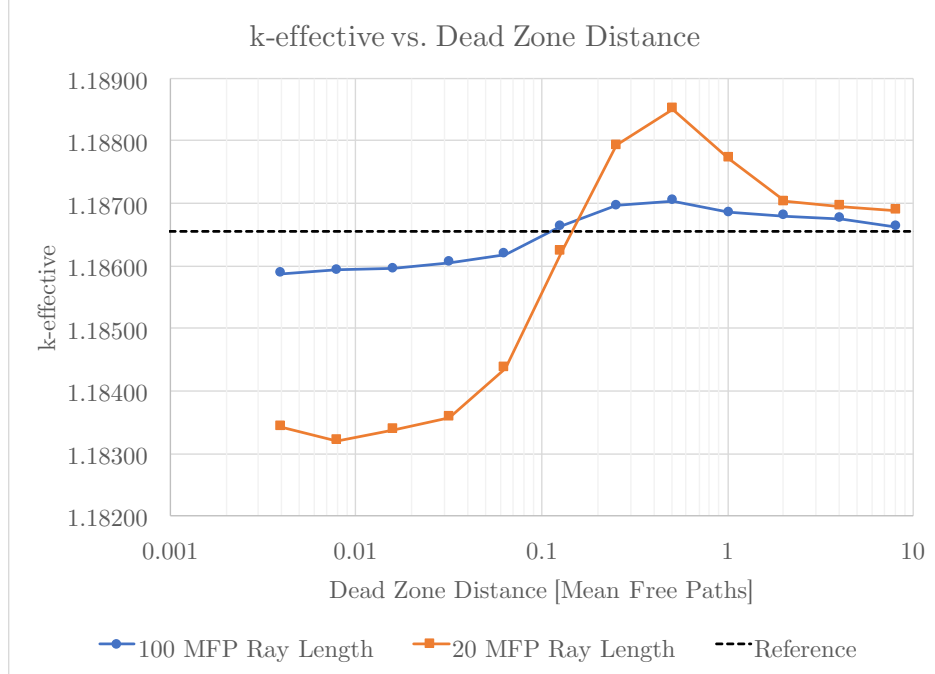
The convergence criteria (SARE) is the simplest parameter and is typically set somewhere between 0.005 and 0.00075, with stricter (lower) criteria reducing the stochastic noise in the solution. Unlike deterministic computations, a looser convergence criteria does not necessarily suggest that the solution will be biased from the true solution. Rather, a looser convergence

criteria will result in a higher variance in the solution but the distribution of solutions will be centered around the true mean provided good choices of ray distance, dead zone length, and ray density are made so as to avoid significant bias. E.g., ensemble runs performed with loose and tight convergence criteria will produce the same statistical means for unknowns but with higher or lower variances. The precise effects of convergence criteria on the realized variance of unknowns of interest such as hot pin power were demonstrated empirically in the previous section.

The next parameters are the termination distance and dead zone distance for each ray. These parameters affect the bias induced into the computation by each ray. In ARRC, each new ray is uniformly sampled randomly in space and angle, uniformly throughout the volume of the full problem domain. No importance weighting, stratified sampling, or other variance reduction techniques are used when selecting ray starting location or angles – they are selected purely from a uniform distribution in space and angle. When domain decomposition is used, selection is also not biased to enforce an equivalent number of rays must start in each subdomain. As the starting angular flux distribution for each ray is unknown an initial guess must be made. In ARRC, each newly sampled ray begins with an angular flux distribution based on where the ray begins. In particular, the beginning angular flux of the ray is set to be equal to the isotropic source distribution of the FSR the ray starts in divided by  $4\pi$ . However, as this is only a first order approximation, significant error is introduced into the computation if further accommodations are not made. One method to reduce the bias from the starting condition is to simply set the ray length to be very high, so as to minimize the impact of the errors from the beginning portions of the ray. The downside to long ray distances however is that there is still some bias due to the starting portions of rays, and also that very long rays reduce the number of angular samples taken given a fixed runtime.

A better error reduction method is to utilize a dead zone<sup>[1]</sup>, wherein a new ray is tracked along its path and its angular flux updated, but no contribution to the scalar flux tally in each FSR is made. This effectively puts the beginning portion of each ray into “read only” mode, where the angular flux of the ray is continually improved without writing back to the system. After the set dead zone distance has been traversed, the ray begins behaving as normal wherein changes in the angular flux through each FSR are tallied to the scalar flux of that FSR. The net effect is that significantly shorter rays can be used (allowing for more angles to be sampled) while simultaneously reducing the bias and error in the simulation. To simplify the parameterization of the dead zone distance and overall ray distance, distances can be input as mean free path (MFP) lengths, where the mean free path can be automatically determined as the absolute highest (or highest material weighted volume average) for all materials in the problem. ARRC can then convert this MFP parameter into geometric distance. While ray distances can also be input directly in terms of centimeters, usage of MFP settings are a little more universal and intuitive and therefore less likely to need to be tailored for different problem types. In this analysis, the maximum MFP in the benchmark problem was used, which occurred in the fast group of the moderator material, though in problems featuring void-like materials it might be recommended to instead use the overall volume average of the maximum MFPs from each material so as to avoid extremely long and overly conservative ray lengths. The effect between dead zone distance, overall ray distance, and accuracy is examined for the 2D C5G7 benchmark problem, with results shown in [Figure 14](#). This figure shows that longer dead zones improve accuracy and reduce the overall ray length requirements.

In practice, it is found that approximately 2 mean free paths (in this case, 12.56 cm) was an optimal dead zone length. While having a larger dead zone distance results in greater accuracy, it is important to remember that dead zone computation is effectively wasted time as it does not contribute to the scalar flux solution to the problem, so for efficiency purposes the dead zone length should be minimal compared to the termination distance of the ray. The optimal termination distance was found to be 100 mean free paths (in this case, 628 cm), which afforded high accuracy while still allowing for a high number of discrete rays per iteration to be sampled while maintaining efficiency.



**Figure 14:** Eigenvalue errors vs. dead zone distance for two different ray termination distances on an example benchmark problem (2D C5G7). All distances are measured in mean free paths (MFPs) of the fast neutron group in the moderator material of the problem.

The final parameter, the number of random rays to run each power iteration, adjusts the overall ray density of each iteration in the simulation given a fixed ray length. There are a variety of competing effects on error and runtime caused by changing the ray density. The most important effect however is that the ray density directly controls the overall time spent in the inactive portion of the simulation. It is desirable to minimize the amount of time spent in this region and to enter the active region as quickly as possible. If roughly the same number of iterations are required to create a stationary source due only to the apparent dominance ratio of the problem being simulated, then the coarsest configuration possible is advantageous so as to reach a stationary source in the minimum amount of time. However, if too few rays are used, eventually a large number of FSRs will be missed every power iteration which may cause error.

Another important competing effect regarding ray density is that there is some overhead to performing a power iteration (e.g., computing new sources, computing k-effective, etc). Therefore, there may be some performance penalty paid for configurations that require an extremely high number of active power iterations. Additionally, it is important to note that

the thread level parallelism in ARRC is expressed over individual rays, where each thread processes one ray at a time. On many HPC architectures, this necessitates that a sufficient number of rays be run to allow for each thread to have at least one ray to process each power iteration. Having enough rays for each thread to run multiple rays each power iteration can also afford improved load balancing and therefore higher overall computational efficiency.

To assess the relative impacts of these competing effects in ARRC, a variety of ray density configurations were tested on both the 2D and 3D C5G7 problems with both coarse and fine FSR meshes. The results are presented in Tables 6 and 7. The overall trend seen in these tables is that the optimal selection corresponds to when nearly all FSRs in the problem are being hit at least once (on average) every power iteration, with only a fraction of a percent of FSRs being missed each iteration. If a coarser selection is used, both errors and runtime increase. If a denser selection is used, error does continue to decrease very slightly (as measured by APE) but at the cost of significantly increased runtime. When also taking into account the results of Tables 1 and 3, it is clear that reduction in error is more efficiently achieved by refining the FSR mesh and/or tightening the convergence criteria (SARE). Take for instance the lowest runtime 2D coarse mesh case with 650 rays per iteration and 0.358 APE. Increasing the ray count to 2,600 rays per iteration increases the runtime by a factor of 2.04x and reduces the error to 0.312 APE (i.e., a 12% reduction in relative error). Comparatively, Table 1 shows that leaving the ray density the same but reducing the SARE from 0.003 to 0.0015 increased runtime by a factor of 2.59x but reduced the APE to 0.261 (i.e., a 27% reduction in relative error), which is a significantly more time efficient reduction in error. Another example is seen in the 3D C5G7 case, where once fewer than 1% of FSRs were missed every power iteration there was no significant difference in the eigenvalue or pin power errors, with increases in ray density only resulting in longer runtimes. Clear reductions in error in the 3D case were only made when the FSR mesh was refined.

FSR Mesh	Rays/Iter	% FSRs Missed/Iter	Runtime [s]	Runtime [core-hr]	Total Iterations	Inactive Iterations	Active Iterations	Total Integrations	k-eff	k-eff std. dev.	APE	APE std. dev.
Coarse (142,964 FSRs)	81	15.2266%	103.8	1.84	12866	943	11923	2.80E+10	1.19249	0.00017	2.155	0.058
	162	2.8862%	65.2	1.16	6426	976	5450	2.80E+10	1.18687	0.00019	0.442	0.036
	325	0.1216%	53.6	0.95	3720	1111	2609	3.25E+10	1.18676	0.00014	0.369	0.044
	650	0.0003%	52.9	0.94	2298	1018	1280	4.02E+10	1.18673	0.00014	0.358	0.059
	1,300	0.0000%	67.7	1.20	1690	1057	633	5.91E+10	1.18677	0.00014	0.326	0.027
	2,600	0.0000%	108.1	1.92	1452	1138	314	1.02E+11	1.18674	0.00013	0.312	0.038
Fine (591,892 FSRs)	250	4.5299%	290.7	5.17	9209	974	8235	1.42E+11	1.18966	0.00007	0.585	0.035
	500	0.3484%	234.5	4.17	4902	1034	3868	1.51E+11	1.18683	0.00009	0.263	0.034
	1,000	0.0076%	232.0	4.12	2878	998	1880	1.77E+11	1.18663	0.00009	0.239	0.022
	2,000	0.0001%	291.2	5.18	1998	1071	926	2.46E+11	1.18665	0.00010	0.229	0.023
	4,000	0.0000%	439.1	7.81	1587	1128	459	3.91E+11	1.18667	0.00008	0.209	0.018
	8,000	0.0000%	777.9	13.83	1444	1217	227	7.12E+11	1.18662	0.00009	0.205	0.026

**Table 6:** The effects of altering the number of rays used each iteration on the runtime and accuracy of the 2D C5G7 benchmark problem. Results were generated using 25 independent ensemble runs to provide average values as well as standard deviations for eigenvalue and average percent error (APE) of the pin power distribution. All tests were performed on a four socket Intel Xeon E7-8867 v3 node with 64 cores total, and were converged to 0.003 SARE.

To help users determine how to set the ray density to hit nearly all FSRs, the percentage of FSRs missed after the first power iteration is displayed so a user can determine if the computation should be halted to adjust this setting up or down. In the future, an option may be added to ARRC to automatically perform a binary search to find the optimal ray count

FSR Mesh	Rays/Iter	% FSRs Missed/Iter	Runtime [s]	Runtime [core-hr]	Total Iterations	Inactive Iterations	Active Iterations	Total Integrations	k-eff	APE
Coarse (25,216,740 FSRs)	<b>14,625</b>	12.4882%	1554.0	1768.1	4470	1244	3226	2.03E+12	1.08028	1.605
	<b>29,250</b>	1.8933%	969.9	1103.5	2556	1056	1500	2.32E+12	1.07770	0.321
	<b>58,500</b>	0.0488%	810.8	922.5	1833	1108	725	3.32E+12	1.07760	0.263
	<b>117,000</b>	0.0002%	900.9	1025.1	1543	1187	356	5.60E+12	1.07768	0.263
	<b>234,000</b>	0.0000%	1317.4	1498.9	1525	1350	175	1.11E+13	1.07766	0.273
	<b>468,000</b>	0.0000%	2561.4	2914.3	1790	1704	86	2.60E+13	1.07761	0.282
Fine (119,205,048 FSRs)	<b>138,250</b>	0.0601%	2732.3	6217.5	2400	1478	922	2.09E+13	1.07772	0.154
	<b>276,500</b>	0.0009%	2575.0	5859.6	1746	1295	451	3.04E+13	1.07768	0.151
	<b>553,000</b>	0.0000%	3410.9	7761.7	1558	1336	222	5.42E+13	1.07766	0.138

**Table 7:** The effects of altering the number of rays used each iteration on the runtime and accuracy of the 3D C5G7 benchmark problem. Results were generated using only a single simulation for each configuration, so uncertainty statistics for eigenvalue and APE are not presented. All tests were performed on the IBM Blue Gene/Q supercomputer *Mira*[33], using 256 nodes for the coarse mesh configurations and 512 nodes for the fine mesh configurations. All tests were converged to 0.005 SARE.

over the course of the first several power iterations of a simulation, in effect automating this process for the user.

In summary, this section has demonstrated how the parameters affecting simulation fidelity are selected optimally by a user of ARRC. Converge criteria reflects the amount of variance in all unknowns. Dead zone distance and ray termination distance reflect the accuracy of the simulation, with a heuristic approach of 2 MFP for the dead zone and 100 MFP for the termination length likely being optimal for a variety of problems as MFP can optionally be set automatically based on the cross section data for each new problem. Finally, it is generally optimal to run the fewest number of rays each power iteration as is possible while still hitting nearly all FSRs in the problem domain at least once each iteration.

## 5. Performance and Parallel Scaling

In this section parallel performance aspects of ARRC are tested. First, the application’s SIMD vector parallelism will be tested by measuring performance while altering the width of the inner loop of the computation in the form of changing the number of neutron energy groups tracked. Then, shared memory threaded parallel scaling will be investigated by measuring speedup on a 44 cores Xeon node. Next, the domain decomposed parallelism strategy deployed in ARRC is discussed including various methods of load balancing. Finally, a number of 2D and large 3D problems are run on the Blue Gene/Q supercomputer *Mira*[33] and strong scaling results are presented.

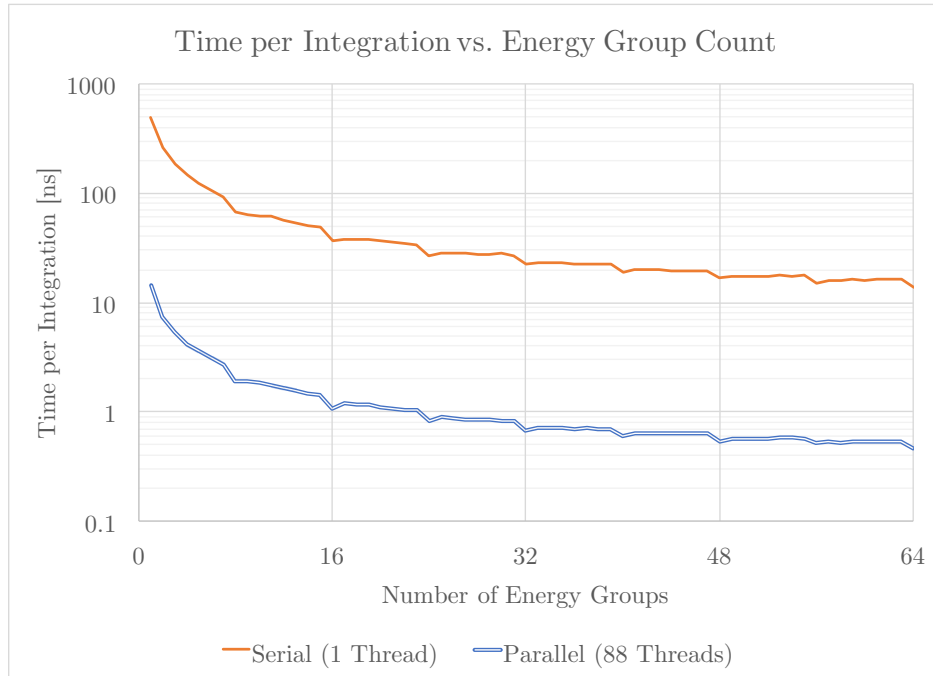
### 5.1. Effects of Energy Group Count

Recent advances in high performance computing (HPC) architectures have been made possible in large part due to the introduction and rapid widening of Single Instruction Multiple Data (SIMD) vector computation units[61, 62]. Currently, most HPC architectures feature SIMD units that are at least 8 single precision floating point values wide, with some as wide as 32 wide as in graphics processing unit (GPU) architectures. This trend in hardware is predicted to continue into the Exascale era of supercomputing, due to the ability for vector units to greatly increase the amount of floating point operations per second



(FLOPS) that a CPU can perform with only modest increases in power and die space usage[36, 61, 62, 63]. The net effect for application developers is that the floating point capabilities of new supercomputers rely heavily on the ability for an application to take advantage of SIMD vector units. For an application to fully take advantage of the resources of new architectures, the application must be vectorizable. 3D MOC can be expressed in a SIMD friendly manner, thanks to the inner loop of the program taking the form of a significant amount of floating point work that is done across all energy groups of the computation independently[27]. This is highly advantageous as a full core computation will have on the order of a hundred energy groups, allowing for a high vectorization efficiency.

To determine how well ARRC is likely to perform on next generation supercomputers with wide vector architectures, a study of the impact of the number of energy groups (i.e., the SIMD vector width) was performed as shown in Figure 15. The study was performed in both serial and parallel configurations using a dual socket Intel Broadwell-EP Xeon E5-2699 v4 node with 44 CPUs (88 threads) total. The test problem used is an example 3D reactor assembly problem, which features a 17x17 reflective lattice of heterogeneous fuel pins. Each fuel pin features discrete fuel, gap, cladding, and moderator regions. The fuel pins are then further subdivided into flat source regions of four quadrants and three radial rings each in the fuel and moderator regions. The axial length of the problem is subdivided into 300 zones. This results in a total of 2,774,400 FSRs, which represents an extremely large problem size for a single node to handle. The large problem size was selected to represent a “worst case” performance scenario, as smaller problems are likely to have increased performance due to lower memory usage and therefore higher cache efficiency.



**Figure 15:** Time per integration [ns] (lower is better) for various numbers of energy groups as measured in serial and parallel on a dual socket Intel Broadwell-EP Xeon E5-2699 v4 node. The SIMD vector width of the machine (8) corresponds to the periodic performance improvements in this plot.

The results in Figure 15 show that ARRC is capable of extreme performance gains

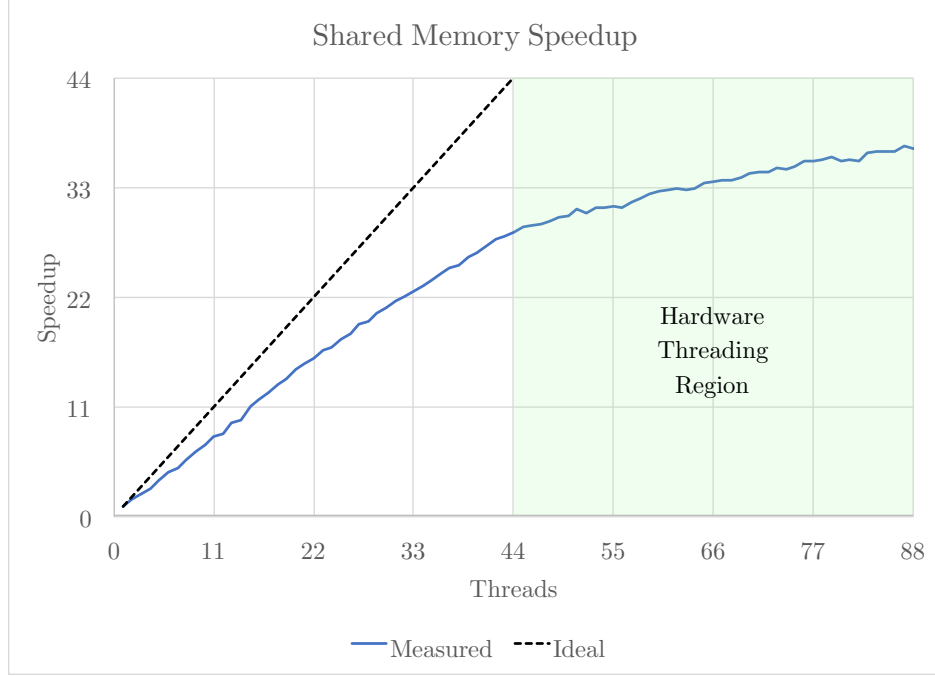
as the number of energy groups increase due to successful vectorization of the inner loop of the application. These SIMD vectorization performance gains are further enhanced by amortization of ray tracing costs over more energy groups. These results suggest that ARRC will be capable of taking full advantage of the performance offered by next generation HPC architectures. It also suggests that the performance results shown in Tables 1 and 3 are likely to improve in terms of efficiency when running on larger reactor problems featuring significantly more than 7 energy groups. For instance, Figure 15 indicates that if 64 group data were to be used in 3D C5G7, ARRC’s time per integration would improve by a factor of 5.8x.

It is also important to note that the 7 energy group decomposition used in the C5G7 benchmarks is a particularly bad choice in terms of performance efficiency, as SIMD vector widths are usually a power of 2. The Intel Broadwell-EP Xeon E5-2699 v4 CPU these tests were performed on features a SIMD vector width of 8. This vector width is clearly visible in Figure 15, where energy group configurations that are even multiples of 8 show sudden performance improvements. These results agree well with previous work done in the SimpleMOC and SimpleMOC-kernel mini-applications (in terms of the impacts of energy group counts and their interplay with SIMD vector widths seen in Figure 15) that offered performance predictions for full scale 3D MOC applications like ARRC[27, 64].

## 5.2. Shared Memory Parallel Scaling

Another significant aspect of high performance computing is the ability for an application to efficiently utilize all CPU cores of a given compute node. Next generation HPC architectures like the Intel Xeon Phi feature several hundred threads per node with more expected each generation. To test ARRC’s shared memory threading capabilities, a scaling test was run on the same 3D heterogenous assembly problem with 64 energy groups as was used in the previous subsection. The problem required approximately 2.7 GB of memory. The tests were performed on a dual socket Intel Broadwell-EP Xeon E5-2699 v4 node with 44 CPUs (88 threads) total. The number of threads was varied from 1 to 88 and the runtime was measured for each configuration. Note that there are only 44 full CPU cores present on the node, with runs over 44 threads making use of the architecture’s hardware threads. Hardware threads are not expected to scale as efficiently as full CPU cores.

The results of this test, shown in Figure 16, show the speedup versus the number of threads used, where speedup is the ratio of runtime of the single threaded case to the  $n$  threaded case. The ideal case (e.g., where  $n$  threads results in a speedup of  $n$ ) is also plotted out to 44 threads. Compared to the serial case, Figure 16 shows that ARRC achieves a speedup of 28.5x when using 44 cores, which is an efficiency of 64.8%. When hardware threads are included in this computation ARRC achieves a speedup of 37x, resulting in an improved efficiency of 84.1%. Shared memory scaling losses are often unavoidable to some degree due to saturation of shared resources (such as bandwidth to main memory, shared cache, etc). However, 84.1% is of similar or better efficiency compared with other HPC neutronics applications[65, 66, 67]. These results show that ARRC is capable of efficient shared memory scaling.



**Figure 16:** Speedup (higher is better) for various numbers of threads as measured in serial and parallel on a dual socket Intel Broadwell-EP Xeon E5-2699 v4 node with 44 cores and 88 threads. Note that the shaded area on the right of the plot is the hardware threading region where scaling is expected to be less efficient.

### 5.3. Domain Decomposed Load Balancing

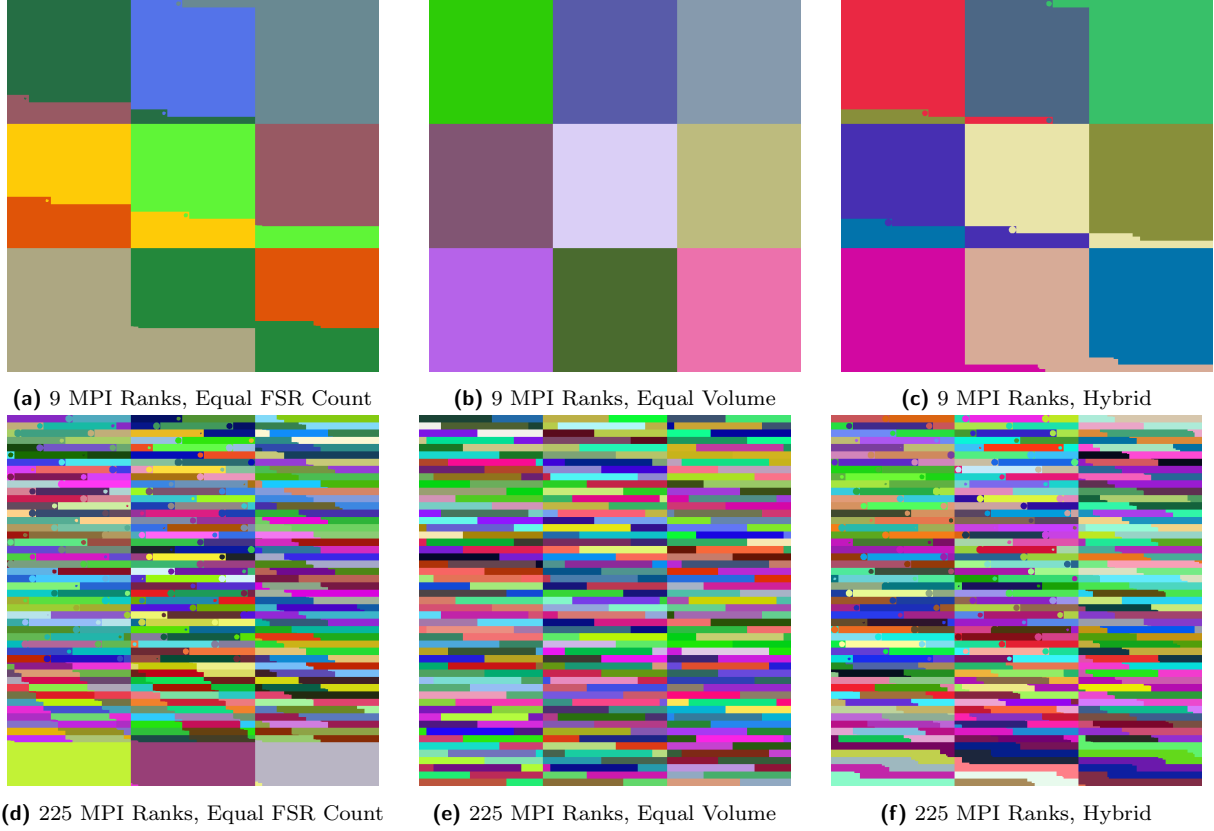
For domain decomposed applications it is always important to account for variable computational loads across domains and ensure that these loads are as even as possible. Failure to account for load imbalances can severely impact performance and limit the parallel scalability of an application. In the context of a neutronics simulation there are significant issues that cause load imbalancing issues. For example, in the C5G7 benchmark, the heterogeneous geometry of the problem allows for a very coarse FSR mesh to be used towards the edges of the moderator regions of the problem where the flux gradients are very low, as seen in Figure 4. This results in having very few FSRs in large spatial regions of the problems compared to the small and tightly packed FSR discretization in the fueled region and near the fuel-moderator interface, as shown in Figure 3. This is important as a naive geometric discretization into equally sized cubes or rectangular prisms may result in extremely poor load balancing.

The central load balancing issue in this context is that the amount of computational work that a subregion must perform in a given power iteration is directly proportional to both the total volume of the subregion and the number of FSRs in that subregion. In a given power iteration, the random rays will travel throughout the entire volume of the problem in an even fashion on average. Therefore, the total distance of rays that any given subdomain of the problem will need to process is proportional to the volume of that subdomain. Additionally, the amount of work it will take to process a set distance of rays through a subdomain is proportional to the number of FSRs in that subdomain. For example, a ray travelling through a subdomain with a very fine FSR decomposition will require many ray traces and angular flux integrations due to the high occurrence of FSR surfaces. Therefore, in aggregate,

the computational load a subdomain experiences in each power iteration is going to be proportional to both the volume and the number of FSRs of the subdomain, as defined in Equation 3.

$$\text{Load} \propto \text{Volume} \times \text{Number of FSRs} \quad (3)$$

Take for example a discretization of the 2D C5G7 geometry from Figure 3 across 9 MPI ranks. In an even volume distribution, 4 MPI ranks will be assigned to the fuel assemblies while 5 MPI ranks will be assigned to the moderator regions as shown in Figure 17. As the fueled regions have significantly more FSRs, rays traversing through those subdomains will require much more computational work which slows those MPI ranks down. The net result is that the moderator MPI ranks will finish their work very quickly and have to wait at line 15 of Algorithm 1 while the fuel assembly MPI ranks finish processing rays. This means that 5 MPI ranks (more than half the computational resources) are often wasted while waiting for 4 fuel region MPI ranks to finish.



**Figure 17:** Domain decomposition patterns for 2D C5G7 using different methods and numbers of ranks.

Instead of an equal volume distribution there is also the option for an equal FSR distribution, wherein each MPI rank receives the same number of FSRs. In our example 2D C5G7 discretization with 9 MPI ranks, this would reduce the MPI ranks used in the moderator, as shown in Figure 17. However, this discretization is also problematic as the number of rays a subdomain will need to process in each subiteration is determined not by the number of FSRs it has, but the volume of the subdomain. I.e., the probability of a ray residing in any

given subdomain is equal to the volume of that subdomain divided by the total volume of the system. Even though rays residing in a low density subdomain like the moderator will need fewer intersections to process, the MPI rank will have vastly more rays to process which may result in equally poor load balancing per Equation 3.

A better strategy is to evenly weight volume and the number of FSRs together when deciding on a decomposition routine so as to ensure all subdomains have a roughly equivalent load per Equation 3. ARRC implements this strategy by selecting contiguous blocks of FSRs to assign to each MPI rank wherein all subdomains have an equal “load” value  $L_d$ , as defined in Equation 5 where  $N_d$  is the number of FSRs in the subdomain  $d$ , and  $V_i$  is the normalized volume of each FSR in the subdomain. The normalized volume is computed as in Equation 4, where  $V_{FSR}$  is the actual volume of the FSR and  $N$  is the total number of FSRs in the problem. This normalization enforces that the number of FSRs in a subdomain and the volume of the subdomain have equal weight in the decomposition strategy. The target  $L_d$  for each subdomain is computed using Equation 6. Once this target has been set, contiguous FSRs are added to an MPI rank’s subdomain until  $L_d$  becomes greater than the target.

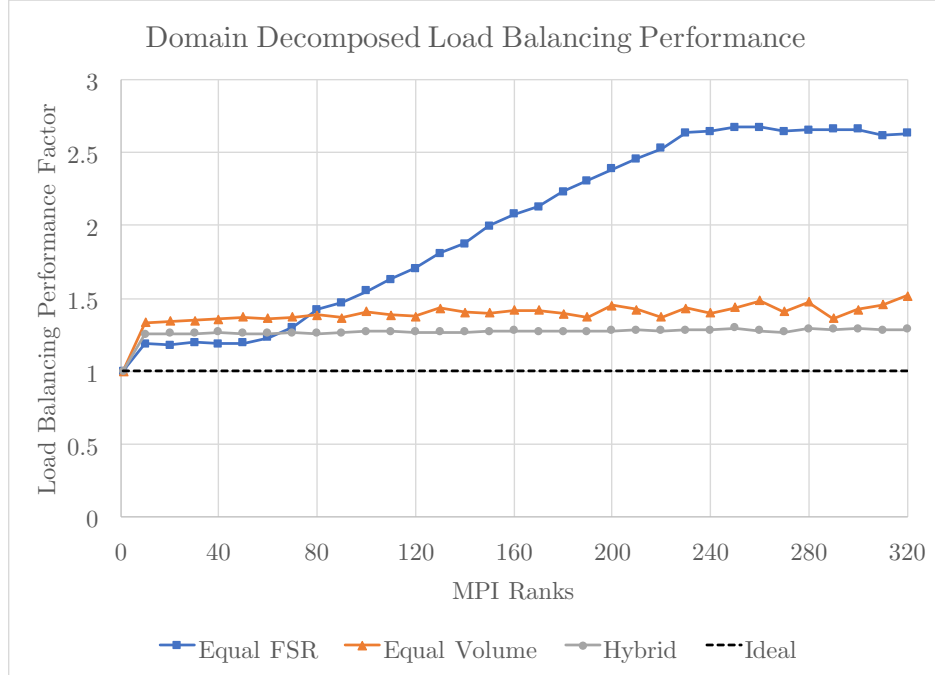
$$V_i = V_{FSR} \frac{N}{V_{total}} \quad (4)$$

$$L_d = N_d + \sum_i^{N_d} V_i \quad (5)$$

$$\text{Target } L_d = \frac{2N}{\# \text{ MPI Ranks}} \quad (6)$$

Improved load balancing results in significant improvements to runtime performance. To test the actual performance, the 2D C5G7 problem was run using all three discretization techniques: the equal FSR count method, the equal volume method, and the hybrid method where FSRs and volume are weighted equally. Several example decompositions using the three methods are shown in Figure 17. To measure performance, the “load balancing performance factor” for each configuration is collected, which is calculated as being the ratio of the maximum number of integrations performed by any single MPI rank to the average number of integrations performed by all MPI ranks. An ideal load balancing scheme will result in a factor of 1.0, with less optimal load balancing schemes having ratios greater than 1.0. For example, a load balancing factor of 2.0 would indicate that a single node is doing twice the work of the average, meaning that the simulation will take twice as much time as compared to the ideal case. The results of this test for the three configurations are given in Figure 18, which shows that the hybrid method is able to achieve significant improvements compared to the others, particularly at scale. At 320 MPI ranks, the hybrid method is 2.05x better than the equal FSR decomposition and 1.18x better than the equal volume decomposition. The poor performance of the equal FSR decomposition at scale is caused by the FSR count being low enough that a single MPI rank is responsible for a large portion of the outer moderator where the mesh is extremely sparse (as seen in Figure 3), causing an extreme volume imbalance. This imbalance can be seen clearly in Figure 17d, where three MPI ranks are responsible for a very large portion of the entire domain. Eventually, once the FSR count per MPI rank becomes low enough that many FSRs are being used to handle this low mesh

density moderator region, the load balancing degradation plateaus. Overall, the results of this test show that the hybrid method was found to be most efficient, so it was therefore used in the strong scaling studies performed in the following section.



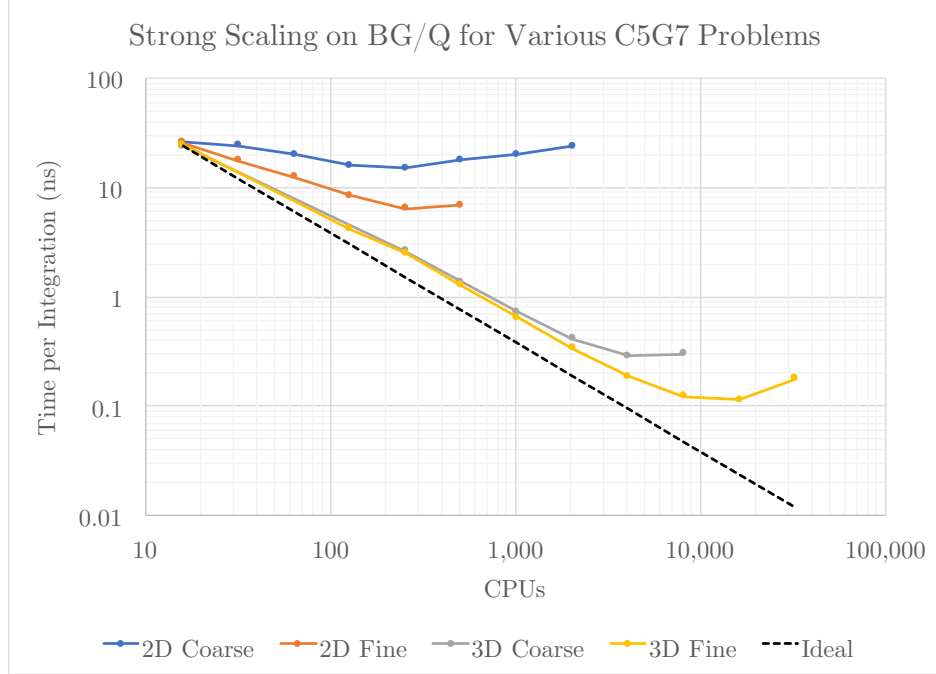
**Figure 18:** Load balancing performance vs. the number of MPI ranks used for the 2D C5G7 problem with the fine mesh configuration. Ideal performance is 1.0, indicating all nodes are doing the same amount of work, with higher ratios indicating poor load balancing.

#### 5.4. Domain Decomposed Parallel Scaling

To test the performance of our hybrid parallel decomposition strategy, a strong scaling study using various configurations of the 2D C5G7 and 3D C5G7 “Rodded B” benchmark problems was performed on the IBM Blue Gene/Q supercomputer *Mira*[33] up to 2048 nodes (32,768 CPUs, 131,072 threads). Results are shown in Figure 19, where performance is measured as the *time per integration* which is computed as the total runtime of the program divided by the total number of inner loop evaluations performed over the course of the computation. Figure 19 shows good scaling out to 100’s of CPUs for the 2D problem and to around 10,000 CPUs for the 3D problem. Scalability for these problems is ultimately limited by on node load balancing, as when there are fewer than 5-10 rays per thread available per power iteration dynamic load balancing can no longer compensate for the disparities in ray lengths through the domain. This effect grows progressively worse towards the end of each power iteration when many rays have terminated and only a few are left to complete per communication sweep.

These results are highly encouraging as they suggest a full core 3D high fidelity reactor simulation, which is orders of magnitude larger than the C5G7 3D problem, may scale well up to even higher node counts. It is also important to note that a more realistic full core computation such as BEAVRS would likely use on the order of 100 energy groups, which would afford nodes significantly more work between communication rounds and would therefore





**Figure 19:** Strong scaling for various C5G7 benchmark configurations on the IBM Blue Gene/Q Supercomputer *Mira*[\[33\]](#).

suggest improvements to maximum speedup beyond what was measured in our 7 group C5G7 test problems.

## 6. Conclusions

We have presented a new 3D nuclear reactor simulation program, ARRC, that is one of the first implementations of a newly developed neutron transport algorithm known as The Random Ray Method (TRRM). While previous work on the topic of TRRM detailed the new algorithm and its mathematical derivations[\[1\]](#), the method has to date only been tested on relatively small benchmark problems. In this study, a highly parallel domain decomposed application was presented that implements TRRM, and supercomputer simulation results for a variety of large 3D benchmark problems were presented. Additionally, a simpler and more optimal convergence scheme (using Shannon Entropy for stationary source detection and the “SARE” convergence criteria) was developed and implemented in ARRC. This effort was done with the goal of better understanding the parallel performance and accuracy characteristics of the new method when applied to large 3D reactor physics problems.

We have shown that the new program, ARRC, is capable of providing similar or better performance and accuracy as compared to deterministic MOC on both 2D and 3D benchmark problems. ARRC was also shown to be capable of efficient parallel scaling due to its extremely low memory requirements and three-tiered parallelism architecture featuring vectorized SIMD computation and task based dynamic load balancing at the node scale, as well as domain decomposed parallelism for use on clusters or supercomputers.

ARRC is therefore shown to be a useful tool for reactor simulation, offering the following advantages over existing applications:

- has the ability to handle arbitrary 3D constructive solid geometries, with no axial homogeneity or basis of symmetry required
- provides extreme improvements in memory efficiency, accurately resolving the 3D C5G7 “Rodded B” benchmark using only 3.5 GB of memory
- makes full use of the SIMD vector floating point units on modern architectures, as evidenced by [Figure 15](#)
- offers efficient on-node shared memory scaling to a high number of CPU cores, as shown in [Figure 16](#)
- offers an accurate estimate of the variance of the generated solution
- uses an optimized load balancing approach for efficient domain decomposed distributed memory scaling on many node supercomputer architectures which is over 2x better than an equal FSR count decomposition approach
- has acceptably low inter-node communication costs affording good scalability on many node supercomputing architectures
- is likely to map very well onto next generation supercomputing architectures due to its ability to be domain decomposed at the distributed memory level, its ability for dynamic task based parallelism at the shared memory level, and for its ability to vectorize at the thread instruction level

This in-depth performance analysis provides a foundational understanding of ARRC on cutting edge high performance computing architectures on large nuclear reactor benchmark problems. Future development of ARRC is likely to be focused on accelerating numerical convergence by implementing a coarse mesh finite difference (CMFD) operator[57] or another acceleration technique. Additionally, improved non-blocking communication schemes for domain decomposed ray exchange will also be pursued in order to reduce the communication costs in ARRC.

## Acknowledgement

Special thanks to the entire Computational Reactor Physics Group at MIT for their help with this work. Particular thanks to Derek Gaston for acting as a sounding board while developing ARRC and this paper.

We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This work was also supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory (“Argonne”) under Contract DE-AC02-06CH11357 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf,

a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

## References

- [1] J. R. Tramm, K. S. Smith, B. Forget, A. R. Siegel, [The Random Ray Method for neutral particle transport](#), Journal of Computational Physics 342 (2017) 229 – 252. doi:<https://doi.org/10.1016/j.jcp.2017.04.038>. URL <http://www.sciencedirect.com/science/article/pii/S0021999117303170>
- [2] K. Smith, [Reactor core methods](#), in: M&C 2003, Gatlinburg, Tennessee, 2003, plenary Presentation. URL <https://www.oecd-nea.org/dbprog/documents/MC03Smith.pdf>
- [3] J. Duderstadt, L. Hamilton, Nuclear Reactor Analysis, Wiley, 1976.
- [4] A. Hebert, Applied Reactor Physics, Presses Internationales Polytechnique, 2009.
- [5] E. Shermon, M. Smith, A. Marin-Lafleche, [PROTEUS-SN user manual](#), Tech. Rep. ANL/NE-14/6 (2014). URL <http://www.ipd.anl.gov/anlpubs/2014/08/79170.pdf>
- [6] Idaho National Laboratory: Rattlesnake, <https://moose.inl.gov/rattlesnake/SitePages/Home.aspx> (2017).
- [7] R. E. Alcouffe, R. S. Baker, J. A. Dahl, S. A. Turner, R. Ward, PARTISN: A time-dependent, parallel neutral particle transport code system, Tech. Rep. LA-UR-05-3925 (2005).
- [8] J. Briesmeister, [MCNP: a general Monte Carlo code for neutron and photon transport. version 3a. revision 2](#), Tech. Rep. LA-7396-M-Rev.2 (Sep 1986). URL [http://www.iaea.org/inis/collection/NCLCollectionStore/\\_Public/18/044/18044302.pdf](http://www.iaea.org/inis/collection/NCLCollectionStore/_Public/18/044/18044302.pdf)
- [9] P. K. Romano, B. Forget, The OpenMC Monte Carlo particle transport code, Annals of Nuclear Energy 51 (C) (2013) 274–281.
- [10] K. Wang, Z. Li, D. She, J. Liang, Q. Xu, Y. Qiu, J. Yu, [RMC – a Monte Carlo code for reactor physics analysis](#), in: SNA + MC 2013 - Joint International Conference on Supercomputing in Nuclear Applications + Monte Carlo, 2013. URL <https://doi.org/10.1051/snamc/201406020>
- [11] J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, T. Kaltiaisenaho, The Serpent Monte Carlo code: Status, development and applications in 2013, Annals of Nuclear Energy 82 (2015) 142–150. doi:[10.1016/j.anucene.2014.08.024](https://doi.org/10.1016/j.anucene.2014.08.024).

- [12] J. Nimal, T. Vergnaud, [TRIPOLI: A general Monte Carlo code, present state and future prospects](#), Progress in Nuclear Energy 24 (1) (1990) 195 – 200. doi:[http://dx.doi.org/10.1016/0149-1970\(90\)90036-5](http://dx.doi.org/10.1016/0149-1970(90)90036-5).  
URL <http://www.sciencedirect.com/science/article/pii/0149197090900365>
- [13] W. Boyd, S. Shaner, L. Li, B. Forget, K. Smith, [The OpenMOC Method of Characteristics neutral particle transport code](#), Annals of Nuclear Energy 68 (2014) 43–52. doi:[10.1016/j.anucene.2013.12.012](http://dx.doi.org/10.1016/j.anucene.2013.12.012).  
URL <http://dx.doi.org/10.1016/j.anucene.2013.12.012>
- [14] B. Kochunas, B. Collins, D. Jabaay, T. J. Downar, W. R. Martin, Overview of development and design of MPACT : Michigan Parallel Characteristics Transport Code, in: M&C 2013: International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, 2013.
- [15] J. Rhodes, K. Smith, D. Lee, [CASMO-5 development and applications](#), in: PHYSOR 2006: ANS Topical Meeting on Reactor Physics, 2006.  
URL [https://www.researchgate.net/publication/228528270\\_CASMO-5\\_development\\_and\\_applications](https://www.researchgate.net/publication/228528270_CASMO-5_development_and_applications)
- [16] A. Yamamoto, A. Giho, Y. Kato, T. Endo, [GENESIS : A three-dimensional heterogeneous transport solver based on the legendre polynomial expansion of angular flux method](#), Nuclear Science and Engineering 186 (April) (2017) 1–22. doi:[10.1080/00295639.2016.1273002](http://dx.doi.org/10.1080/00295639.2016.1273002).  
URL <http://dx.doi.org/10.1080/00295639.2016.1273002>
- [17] Y. Zheng, S. Choi, D. Lee, [A new approach to three-dimensional neutron transport solution based on the Method of Characteristics and linear axial approximation](#), Journal of Computational Physics (August). doi:[10.1016/j.jcp.2017.08.026](http://dx.doi.org/10.1016/j.jcp.2017.08.026).  
URL <http://linkinghub.elsevier.com/retrieve/pii/S0021999117305971>
- [18] R. Sanchez, Prospects in deterministic three-dimensional whole-core transport calculations, Nuclear Engineering and Technology 44 (5) (2012) 113–150. doi:[10.5516/NET.01.2012.501](http://dx.doi.org/10.5516/NET.01.2012.501).
- [19] D. Sciannandrone, S. Santandrea, Tracking strategies in 3D axial geometries for a MOC solver, in: SNA + MC 2013 - Joint International Conference on Supercomputing in Nuclear Applications + Monte Carlo, 2013.
- [20] H. Prabha, G. Marleau, [Computation of 3D neutron fluxes in one pin hexagonal cell](#), Annals of Nuclear Energy 56 (2013) 1–6. doi:[10.1016/j.anucene.2012.12.023](http://dx.doi.org/10.1016/j.anucene.2012.12.023).  
URL <http://dx.doi.org/10.1016/j.anucene.2012.12.023>
- [21] H. Prabha, G. Marleau, A. Hébert, [Tracking algorithms for multi-hexagonal assemblies \(2D and 3D\)](#), Annals of Nuclear Energy 69 (2014) 175–182. doi:[10.1016/j.anucene.2014.01.018](http://dx.doi.org/10.1016/j.anucene.2014.01.018).  
URL <http://linkinghub.elsevier.com/retrieve/pii/S0306454914000267>

- [22] M. Eklund, M. Alamaniotis, H. Hernandez, T. Jevremovic, Method of Characteristics - a review with applications to science and nuclear engineering computation, *Progress in Nuclear Energy* 85 (2015) 548–567. doi:[10.1016/j.pnucene.2015.05.002](https://doi.org/10.1016/j.pnucene.2015.05.002).
- [23] G. Grassi, A nonlinear space-angle multigrid acceleration for the Method of Characteristics in unstructured meshes, *Nuclear Science and Engineering* 155 (2007) 208–222.
- [24] B. Kochunas, T. Downar, S. Mohamed, J. Thomas, Improved parallelization of the modular ray tracing in the Method of Characteristics code DeCART, in: *Proceedings of the Joint International Topical Meeting on Mathematics and Computation and Supercomputing in Nuclear Applications (M&C+SNA 2007)*, 2007.
- [25] B. Kochunas, A hybrid parallel algorithm for the 3-D Method of Characteristics solution of the Boltzmann Transport Equation on high performance computing clusters, Ph.D. Thesis, University of Michigan, Department of Nuclear Engineering and Radiological Sciences (2013).
- [26] W. Boyd, [Massively parallel algorithms for Method of Characteristics neutral particle transport on shared memory computer architectures](#), M.S. Thesis, Massachusetts Institute of Technology, Department of Nuclear Science and Engineering (2014).  
URL <http://hdl.handle.net/1721.1/87494>
- [27] J. R. Tramm, G. Gunow, T. He, K. S. Smith, B. Forget, A. R. Siegel, A task-based parallelism and vectorized approach to 3D Method of Characteristics (MOC) reactor simulation for high performance computing architectures, *Computer Physics Communications* 202 (2016) 141 – 150. doi:<http://dx.doi.org/10.1016/j.cpc.2016.01.007>.
- [28] K. Smith, B. Forget, Challenges in the development of high-fidelity LWR core neutronics tools, in: *M&C 2013: International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, 2013, pp. 1809–1825.
- [29] W. S. Yang, M. A. Smith, C. H. Lee, A. Wollaber, D. Kaushik, A. S. Mohamed, [Neutronics modeling and simulation of SHARP for fast reactor analysis](#), *Nuclear Engineering and Technology* 42 (5) (2010) 520–545. doi:[10.5516/NET.2010.42.5.520](https://doi.org/10.5516/NET.2010.42.5.520).  
URL <https://www.kns.org/jknsfile/v42/JK0420520.pdf>
- [30] G. Gunow, J. Tramm, B. Forget, K. Smith, T. He, SimpleMOC – a performance abstraction for 3D MOC, in: *ANS & M&C 2015 - Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method*, 2015.
- [31] G. Gunow, S. Shaner, W. Boyd, B. Forget, K. Smith, Accuracy and performance of 3D MOC for full-core PWR problems, in: *M&C 2017 - International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, 2017.
- [32] N. Horelik, B. Herman, B. Forget, K. Smith, Benchmark for evaluation and validation of reactor simulations (BEAVRS), in: *M&C 2013 – International Conference on*

Mathematics and Computational Methods Applied to Nuclear Science & Engineering, 2013.

- [33] Argonne Leadership Computing Facility: Mira, <https://www.alcf.anl.gov/mira> (2017).
- [34] N. Attig, P. Gibbon, T. Lippert, Trends in supercomputing: The European path to exascale, *Computer Physics Communications* 182 (9) (2011) 2041–2046.
- [35] C. Engelmann, [Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale](#), *Future Generation Computer Systems* 30 (Supplement C) (2014) 59 – 65, special Issue on Extreme Scale Parallel Architectures and Systems, *Cryptography in Cloud Computing and Recent Advances in Parallel and Distributed Systems*, ICPADS 2012 Selected Papers. doi:<https://doi.org/10.1016/j.future.2013.04.014>.  
URL <http://www.sciencedirect.com/science/article/pii/S0167739X13000745>
- [36] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, A. Ramirez, [The low power architecture approach towards exascale computing](#), *Journal of Computational Science* 4 (6) (2013) 439 – 443, *scalable Algorithms for Large-Scale Systems Workshop (ScalA2011)*, *Supercomputing 2011*. doi:<https://doi.org/10.1016/j.jocs.2013.01.002>.  
URL <http://www.sciencedirect.com/science/article/pii/S1877750313000148>
- [37] B. Kochunas, T. J. Downar, Parallel 3-D Method of Characteristics in MPACT, in: *SNA + MC 2013 - Joint International Conference on Supercomputing in Nuclear Applications + Monte Carlo*, 2013, pp. 54–65.
- [38] P. K. Romano, A. R. Siegel, B. Forget, K. Smith, [Data decomposition of Monte Carlo particle transport simulations via tally servers](#), *Journal of Computational Physics* 252 (2013) 20 – 36. doi:<http://dx.doi.org/10.1016/j.jcp.2013.06.011>.  
URL <http://www.sciencedirect.com/science/article/pii/S002199911300435X>
- [39] J. E. Hoogenboom, B. Petrovic, W. R. Martin, Present status and extensions of the Monte Carlo performance benchmark, in: *SNA + MC 2013 - Joint International Conference on Supercomputing in Nuclear Applications + Monte Carlo*, 2013.
- [40] K. G. Felker, A. R. Siegel, S. F. Siegel, [Optimizing memory constrained environments in Monte Carlo nuclear reactor simulations](#), *International Journal of High Performance Computing Applications* 27 (2) (2012) 210–216. doi:[10.1177/1094342012445627](https://doi.org/10.1177/1094342012445627).  
URL <http://hpc.sagepub.com/cgi/doi/10.1177/1094342012445627>
- [41] N. Horelik, A. Siegel, B. Forget, K. Smith, [Monte Carlo domain decomposition for robust nuclear reactor analysis](#), *Parallel Computing* 40 (10) (2014) 646 – 660. doi:<https://doi.org/10.1016/j.parco.2014.10.001>.  
URL <http://www.sciencedirect.com/science/article/pii/S0167819114001240>
- [42] S. Stimpson, B. Collins, B. Kochunas, [Improvement of transport-corrected scattering stability and performance using a Jacobi inscatter algorithm for 2D-MOC](#), *Annals of*



- Nuclear Energy 105 (2017) 1–10. doi:10.1016/j.anucene.2017.02.024.  
URL <http://dx.doi.org/10.1016/j.anucene.2017.02.024>
- [43] C. Tang, S. Zhang, Development and verification of an MOC code employing assembly modular ray tracing and efficient acceleration techniques, Annals of Nuclear Energy 36 (8) (2009) 1013–1020. doi:10.1016/j.anucene.2009.06.007.  
URL <http://dx.doi.org/10.1016/j.anucene.2009.06.007>
- [44] R. M. Ferrer, J. Rhodes, K. Smith, Linear source approximation in CASMO5, in: PHYSOR 2012 – Advances in Reactor Physics – Linking Research, Industry, and Education, 2012.
- [45] A. Siegel, K. Smith, P. Fischer, V. Mahadevan, Analysis of communication costs for domain decomposed Monte Carlo methods in nuclear reactor analysis, Journal of Computational Physics 231 (8) (2012) 3119–3125. doi:10.1016/j.jcp.2011.12.014.  
URL <http://dx.doi.org/10.1016/j.jcp.2011.12.014>
- [46] A. R. Siegel, K. Smith, P. K. Romano, B. Forget, K. Felker, The effect of load imbalances on the performance of Monte Carlo algorithms in LWR analysis, Journal of Computational Physics 235 (2013) 901–911. doi:10.1016/j.jcp.2012.06.012.  
URL <http://dx.doi.org/10.1016/j.jcp.2012.06.012>
- [47] T. A. Brunner, T. J. Urbatsch, T. M. Evans, N. A. Gentile, Comparison of four parallel algorithms for domain decomposed implicit Monte Carlo, Journal of Computational Physics 212 (2) (2006) 527–539. doi:10.1016/j.jcp.2005.07.009.
- [48] H. J. Alme, G. H. Rodrigue, G. B. Zimmerman, Domain decomposition models for parallel Monte Carlo transport, The Journal of Supercomputing 18 (1) (2001) 5–23. doi:10.1023/A:1008196906753.  
URL <https://doi.org/10.1023/A:1008196906753>
- [49] W. Gropp, M. Snir, Programming for exascale computers, Computing in Science and Engineering 15 (6) (2013) 27–35. doi:10.1109/MCSE.2013.96.
- [50] P. K. Romano, Parallel algorithms for Monte Carlo particle transport simulation on exascale computing architectures, PhD Dissertation, Massachusetts Institute of Technology, Department of Nuclear Science and Engineering (2013).  
URL <http://hdl.handle.net/1721.1/80415>
- [51] F. B. Brown, On the use of Shannon Entropy of the fission distribution for assessing convergence of Monte Carlo criticality calculations, in: PHYSOR-2006, ANS Topical Meeting on Reactor Physics, 2006.  
URL <https://www.oecd-neo.org/science/wpncs/sccsa/documents/brown-physor-2006.pdf>
- [52] M. Nowak, J. Miao, E. Dumonteil, B. Forget, A. Onillon, K. S. Smith, A. Zoia, Monte Carlo power iteration: Entropy and spatial correlations, Annals of Nuclear Energy 94 (2016) 856–868. doi:10.1016/j.anucene.2016.05.002.  
URL <http://dx.doi.org/10.1016/j.anucene.2016.05.002>

- [53] E. Dumonteil, F. Malvagi, A. Zoia, A. Mazzolo, D. Artusio, C. Dieudonné, C. De Mulatier, [Particle clustering in Monte Carlo criticality simulations](#), Annals of Nuclear Energy 63 (2014) 612–618. doi:10.1016/j.anucene.2013.09.008. URL <http://dx.doi.org/10.1016/j.anucene.2013.09.008>
- [54] J. Miao, B. Forget, K. Smith, [Analysis of correlations and their impact on convergence rates in Monte Carlo eigenvalue simulations](#), Annals of Nuclear Energy 92 (2016) 81–95. doi:10.1016/j.anucene.2016.01.037. URL <http://dx.doi.org/10.1016/j.anucene.2016.01.037>
- [55] B. R. Herman, B. Forget, K. Smith, P. K. Romano, T. M. Sutton, D. J. Kelly, B. N. Aviles, Analysis of tally correlation in large light water reactors, in: PHYSOR 2014 – The Role of Reactor Physics toward a Sustainable Future, 2014.
- [56] M. A. Smith, E. Lewis, Benchmark on deterministic transport calculations without spatial homogenisation – a 2-D/3-D MOX fuel assembly benchmark, Tech. Rep. ISBN 92-64-02139-6, Nuclear Energy Agency: Organization for Economic Co-Operation and Development (2003).
- [57] K. Smith, [Nodal method storage reduction by non-linear iteration](#), Transactions of the American Nuclear Society (1983) 44. URL [https://inis.iaea.org/search/search.aspx?orig\\_q=RN:15010017](https://inis.iaea.org/search/search.aspx?orig_q=RN:15010017)
- [58] M. A. Smith, E. Lewis, B. Na, Benchmark on deterministic transport calculations without spatial homogenisation – MOX fuel assembly 3-D extension case, Tech. Rep. ISBN 92-64-01069-6, Nuclear Energy Agency: Organization for Economic Co-Operation and Development (2005).
- [59] Oak Ridge National Laboratory: Titan, <https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/> (2017).
- [60] E. D. Walker, [Modeling integral fuel burnable absorbers using the Method of Characteristics](#). Master’s Thesis, University of Tennessee, 2014. URL [http://trace.tennessee.edu/utk\\_gradthes/3191](http://trace.tennessee.edu/utk_gradthes/3191)
- [61] J. Dongarra, S. Tomov, P. Luszczek, J. Kurzak, M. Gates, I. Yamazaki, H. Anzt, A. Haidar, A. Abdelfattah, With extreme computing, the rules have changed, Computing in Science Engineering 19 (3) (2017) 52–62. doi:10.1109/MCSE.2017.48.
- [62] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y. C. Liu, Knights Landing: Second-generation Intel Xeon Phi product, IEEE Micro 36 (2) (2016) 34–46. doi:10.1109/MM.2016.25.
- [63] R. Tian, Simulation at extreme-scale: Co-design thinking and practices, Archives of Computational Methods in Engineering 21 (1) (2014) 39–58. doi:10.1007/s11831-014-9095-y.

- [64] G. Gunow, J. R. Tramm, B. Forget, K. Smith, SimpleMOC – A performance abstraction for 3D MOC, in: ANS&MC 2015 – Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Nashville, 2015.
- [65] J. R. Tramm, A. R. Siegel, Memory bottlenecks and memory contention in multi-core Monte Carlo transport codes, *Annals of Nuclear Energy* 82 (2015) 195 – 202. doi:<http://dx.doi.org/10.1016/j.anucene.2014.08.038>.
- [66] J. R. Tramm, A. R. Siegel, T. Islam, M. Schulz, XSBench - the development and verification of a performance abstraction for Monte Carlo reactor analysis, in: PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future, Kyoto.
- [67] J. R. Tramm, A. R. Siegel, B. Forget, C. Josey, [Performance analysis of a reduced data movement algorithm for neutron cross section data in Monte Carlo simulations](#), in: EASC 2014 - International Conference on Exascale Applications and Software, 2014, pp. 39–56.  
URL [https://link.springer.com/chapter/10.1007/978-3-319-15976-8\\_3](https://link.springer.com/chapter/10.1007/978-3-319-15976-8_3)