PARALLEL MONTE CARLO SYNTHETIC ACCELERATION METHODS FOR DISCRETE TRANSPORT PROBLEMS

by

Stuart R. Slattery

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Nuclear Engineering and Engineering Physics)

at the

UNIVERSITY OF WISCONSIN–MADISON

6 July 2013

Acknowledgments

Great thanks are owed to my advisor and friend Paul Wilson for providing me years of guidance and the opportunity to develop this work. Without him, this work would have never been possible.

Tom Evans is responsible for presenting me with the seeds of this work and for that I thank him. His mentoring over the past few years has been invaluable.

Roger Pawlowski and other members of the Consortium for Advanced Simulation of Light Water Reactors and Trilinos teams as well as many staff members at Oak Ridge National Laboratory have provided tremendous resources for my professional and technical development and have greatly facilitated this work.

This work was performed under appointment to the Nuclear Regulatory Commission Fellowship program at the University of Wisconsin - Madison Department of Engineering Physics.

Contents

PARALLEL MONTE CARLO SYNTHETIC ACCELERATION METHODS FOR DISCRETE TRANSPORT PROBLEMS

Stuart R. Slattery

Under the supervision of Professor Paul P.H. Wilson
At the University of Wisconsin-Madison

Paul P.H. Wilson

Chapter 1

Introduction

In nearly all high-fidelity nuclear reactor simulations, linear and nonlinear transport problems are a primary focus of study. Recent focus on multiple physics systems in the nuclear reactor modeling and simulation community adds a new level of complexity to common linear and nonlinear systems as solution strategies change when they are coupled to other problems (U.S. Department of Energy, 2011). Furthermore, a desire for predictive simulations to enhance the safety and performance of nuclear systems creates a need for extremely high fidelity computations to be performed for these coupled transport systems as a means to capture effects not modeled by coarser methods.

In order to achieve this high fidelity, state-of-the-art computing must be leveraged in a way that is both efficient and considerate of hardware-related issues. As scientific computing moves towards exascale facilities with machines of $O(1,000,000)$ cores already coming on-line, new algorithms to solve these complex problems must be developed to leverage this new hardware (Kogge and Dysart, 2011). Issues such as resiliency to node failure, limited growth of memory available per node, and scaling to large numbers of cores will be pertinent to robust algorithms aimed at this new hardware. Considering these issues, this dissertation develops a massively parallel Monte Carlo method for linear problems and a novel Monte Carlo method to advance solution techniques for nonlinear problems.

We discuss in this chapter physics-based motivation for advancing Monte Carlo solvers and studying their application to transport systems by providing problems of interest in nuclear reactor analysis. Hardware-based motivations are also provided by considering the impact of forthcoming computing architectures. In addition, background on the current solver techniques for multiphysics problems and a brief comparison to the proposed methods is provided to further motivate this work. The organization of the document is then reviewed followed by an explanation of the key outcomes of this work and where in the document results and data supporting those outcomes can be located.

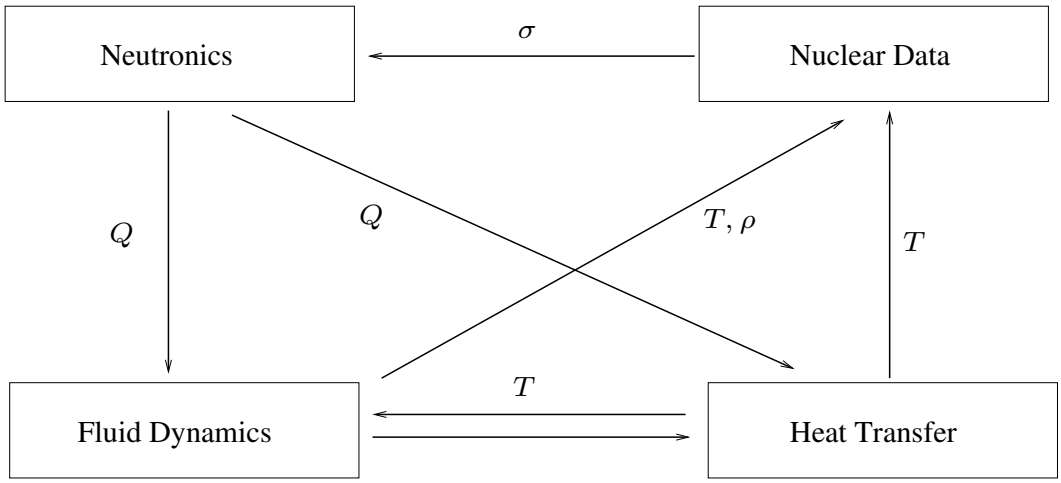

Figure 1.1: **Multiphysics dependency analysis of departure from nucleate boiling.** *A neutronics solution is required to compute power generation in the fuel pins, fluid dynamics is required to characterize boiling and fluid temperature and density, heat transfer is required to compute the fuel and cladding temperature, and the nuclear data modified with the temperature and density data. Strong coupling among the variables creates strong nonlinearities.*

1.1 Physics-Based Motivation

Predictive modeling and simulation capability requires the combination of high fidelity models, high performance computing hardware that can handle the intense computational loads required by these models, and modern algorithms for solving these problems that leverage this high performance hardware. For nuclear reactor analysis, this predictive capability can enable tighter design tolerances for improved thermal performance and efficiency, higher fuel burn-up and therefore reduction in generated waste, and high confidence in accident scenario models. The physics that dominate these types of analysis include neutronics, thermal hydraulics, computational fluid dynamics, and structural mechanics.

Although solution techniques in each of these individual categories has advanced over the last few decades and in fact leveraged modern algorithms and computer architectures, true predictive capability for engineered systems can only be achieved through a coupled, multiple physics analysis where the effects of feedback between physics are modeled. For example, consider the safety analysis of a departure from nucleate boiling scenario in the subchannel of a nuclear fuel assembly. When this event occurs, heat transfer is greatly reduced between the fuel and the coolant due

to the vapor layer generated by boiling, causing the fuel center-line temperature to rapidly rise. To characterize this boiling phenomena and how it affects fuel failure we must consider a neutronics analysis in order to compute power generation in the fuel pins, fluid dynamics analysis to characterize coolant boiling, temperature, and density, solid material heat transfer to characterize fuel and cladding temperature and heat transfer with the coolant, and nuclear data processing to characterize how changing material temperatures and densities changes the cross sections needed for the neutronics calculation. As shown in Figure 1.1, many couplings are required among individual physics components in order to accurately model this situation with each physics generating and receiving many responses. Those variables that are very tightly coupled, such as the temperatures generated by the fluid dynamics and heat transfer components, will have strong nonlinearities in their behavior and would therefore benefit from fully consistent nonlinear solution schemes instead of fixed-point type iterations between physics[1]. Furthermore, the space and time scales over which these effects occur will also vary greatly.

The computational resources required to solve such problems are tremendous. Recent work in modeling coupled fluid flow and solid material heat and mass transfer in a reactor subsystem, similar to the same components of the departure from nucleate boiling example above, was performed as part of analysis of the Department of Energy's Consortium for Advanced Simulation of Light Water Reactors (CASL) modeling and simulation hub. CASL used the Drekar multiphysics code developed at Sandia National Laboratories (Pawlowski et al., 2012) for modeling goals in grid-to-rod-fretting analysis and will use a similar coupled physics structure for future departure from nucleate boiling analysis with comparison to experimental data. Using Drekar, multiphysics simulations have been performed with fully consistent methods for the solution of nonlinear systems using meshes of $O(1 \times 10^9)$ elements leveraging $O(100,000)$ cores on leadership class machines. Neutronics components to be implemented in CASL for multiphysics analysis, such as the Exnihilo radiation transport suite developed at Oak Ridge National Laboratory (Evans et al., 2010), compute trillions of unknowns for full core reactor analysis on $O(1 \times 10^9)$ element meshes and $O(100,000)$ cores as well. Given the large scale and complexity of these problems, if we aim to advance multiphysics solution techniques, then we are motivated to advance the solution of complex and general nonlinear problems exploiting leadership class levels of parallelism.

[1]Fixed-point iterations between physics are commonly referred to as Picard iterations.

1.1.1 Solutions for the SP_N Equations

The neutron transport problem is complicated. Solutions cover a large phase space and the problems of interest are often geometrically complex, very large, or both, requiring tremendous computational resources to generate an adequate solution. Modern deterministic methods for large scale problems are commonly variants on the discrete ordinates (S_N) method (Evans et al., 2010). For fission reactor neutronics simulations, the S_N method requires potentially trillions of unknown angular flux moments to be computed to achieve good accuracy for the responses of interest (Slaybaugh, 2011). Other forms of the transport problem, including the P_N method, take on a simpler form than the more common S_N methods but lack in accuracy when compared while still requiring considerable computational resources for solutions in multiple dimensions.

In the 1960's, Gelbard developed an ad-hoc multidimensional extension of the simple single dimension planar P_N equations that created a system of coupled, diffusion-like equations known as the simplified P_N (SP_N) equations. Up until around the 1990's, the SP_N method was either widely unknown, widely unused, or combination of both even though numerical studies showed promising results with better solutions than diffusion theory and a significant reduction in computational time over more accurate methods such as discrete ordinates. Why did this happen? A significant problem, pointed out by Brantley and Larsen (Brantley and Larsen, 2000), was that little rigor had been applied to the formulation of the SP_N equations since their derivation through primarily heuristic arguments. Instead, studies at that time focused on simply comparing the results of the method to other contemporary transport solution strategies. In addition, many problems of interest from the literature at the time were either solved using nodal-type methods for reactor-sized problems or S_N-type methods for benchmark problems with intricate material configurations and potentially large flux gradients over small spatial domains.

So why reconsider the SP_N equations? Starting in the 1990's and primarily due to Larsen and his colleagues, the SP_N equations have been given a more rigorous treatment with both variational and asymptotic derivations performed as a means of verification. In addition, these equations have been more rigorously studied as solution methods to MOX fuel problems and have been shown to provide accurate solutions. With this mathematical literature to provide a solid numerical footing for the method, we look at its application to today's challenge problems in neutron transport for fission reactor analysis. The reduction in numerical complexity of current deterministic solution methods using the S_N approximation could mean significant

savings in both compute time and memory required. In addition, the characteristics of the solution to the transport problem for a steady state reactor core permit diffusion theory to be used; a staple of the nuclear industry since its inception. Therefore, if diffusion theory is applicable, then finer grained solutions that capture more of the physics contained in the transport equation should be possible with the SP_N method. In doing so, we also expect from the literature to obtain computed responses on the order of accuracy we would expect from an appropriately discretized S_N method at a fraction of the cost.

In order to leverage MCSA as a solution technique for physics problems, its current formulation requires the linear operator and all preconditioners to be explicitly formed. Recent developments in the Exnihilo neutronics package at Oak Ridge National Laboratory have permitted generation of the SP_N system of equations for detailed neutronics models of fission reactor-based systems(Evans, 2013). By fully forming these equations and formulating them as a linear algebra problem instead of using the explicit iterative methods of the past, we now have access to all of the modern advancements in computational linear algebra including Krylov solvers for asymmetric systems and algebraic preconditioning methods. This leads us to then explore the applicability of our work in discrete Monte Carlo methods for linear systems as a possible solution method for the SP_N equations. In addition, solving the SP_N equations in this way also breaks away from the S_N forms of parallelism where spatial parallelism is achieved by an efficient parallel sweep, angular efficiency achieved by pipe-lining, and energy parallelism achieved by decoupling the groups. With the SP_N equations as a full matrix system, we now can parallelize the problem as prescribed by the linear solver, which may be significantly more scalable than current S_N transport practices.

1.1.2 Solutions for the Navier-Stokes Equations

1.2 Hardware-Based Motivation

As leadership class machines move towards the exascale, new algorithms must be developed that leverage their strengths and adapt to their shortcomings. Basic research is required now to advance methods in time for these new machines to become operational. Organized work is already moving forward in this area with the Department of Energy's Advanced Scientific Computing Research office specifically allocating funding for the next several years to research resilient solver technologies

for exascale facilities (U.S. Department of Energy, 2012). Based on the language in this call for proposals, we can identify key issues for which a set of robust, massively parallel Monte Carlo solvers could provide a solution. As machines begin to operate at hundreds of petaflops peak performance and beyond, trends toward reduced energy consumption will require incredibly high levels of concurrency to achieve the desired computation rates. Furthermore, this drop in power consumption will mean increased pressure on memory as memory per node is expected to stagnate while cores per node is expected to increase. As the number of cores increases, their clock speed is expected to stagnate or even decrease to further reduce power consumption and manufacturing costs.

The end result of these hardware changes is that the larger numbers of low-powered processors will be prone to both soft failures such as bit errors in floating point operations and hard failures where the data owned by that processor cannot be recovered. Because these failures are predicted to be common, resilient solver technologies are required to overcome these events. With linear and nonlinear solvers based on Monte Carlo techniques, such issues are alleviated by statistical arguments. In the case of soft failures, isolated floating point errors in Monte Carlo simulation are absorbed within tally statistics while completely losing hardware during a hard failure is manifested as a high variance event where some portion of the Monte Carlo histories are lost. These stochastic methods are a paradigm shift from current deterministic solver techniques that will suffer greatly from the non-deterministic behavior expected from these exascale machines.

In addition to resiliency concerns, the memory restrictions on future hardware will hinder modern solvers that derive their robustness from using large amounts of memory. Stochastic methods that are formulated to use less memory than conventional methods will serve to alleviate some of this pressure. In addition, new parallel strategies that may be implemented with stochastic methods could offer a new avenue for leveraging the expected levels of high concurrency in exascale machines.

1.3 Research Outline

For some time, the particle transport community has been utilizing Monte Carlo methods for the solution of transport problems (Lewis, 1993). The partial differential equation (PDE) community has focused on various deterministic methods for solutions to linear problems (Saad, 2003; Kelley, 1995). In between these two areas are a not widely known group of Monte Carlo methods for solving sparse linear systems (Forsythe

and Leibler, 1950; Hammersley and Handscomb, 1964; Halton, 1962, 1994). In recent years, these methods have been further developed for radiation transport problems in the form of Monte Carlo Synthetic-Acceleration (MCSA) (Evans and Mosher, 2009; Evans et al., 2012) but have yet to be applied to more general sparse linear systems commonly generated by the computational physics community. Compared to other methods in this regime, MCSA offers three attractive qualities; (1) the linear problem operator need not be symmetric or positive-definite, thereby reducing preconditioning complexity, (2) the stochastic nature of the solution method provides a natural solution to the issue of resiliency, and (3) is amenable to parallelization using modern methods developed by the transport community (Wagner et al., 2010). The development of MCSA as a general linear solver and the development of a parallel MCSA method are new and unique features of this work, providing a framework with which other issues such as resiliency may be addressed in the future.

In addition to linear solver advancements, nonlinear solvers may also benefit from a general and parallel MCSA scheme. In the nuclear engineering community, nonlinear problems are often addressed by either linearizing the problem or building a segregated scheme and using traditionally iterative or direct methods to solve the resulting system (Pletcher et al., 1997). In the mathematics community, various Newton methods have been popular (Kelley, 1995). Recently, Jacobian-Free Newton-Krylov (JFNK) schemes (Knoll and Keyes, 2004) have been utilized in multiple physics architectures and advanced single physics codes (Gaston et al., 2009). The benefits of JFNK schemes are that the Jacobian is never formed, simplifying the implementation, and a Krylov solver is leveraged (typically GMRES or Conjugate Gradient), providing excellent convergence properties for well-conditioned and well-scaled systems. However, there are two potential drawbacks to these methods for high fidelity predictive simulations: (1) the Jacobian is approximated by a first-order differencing method on the order of machine precision such that this error can grow beyond that of those in a fine-grained system (Kelley, 1995) and (2) for systems that are not symmetric positive-definite (which will be the case for most multiphysics systems and certainly for most preconditioned systems) the Krylov subspace generated by the GMRES solver may become prohibitively large (Knoll and McHugh, 1995). To address these issues, this work develops a new and novel method for nonlinear systems based on the MCSA method.

The Forward-Automated Newton-MCSA (FANM) method is developed as new nonlinear solution method. The key features of FANM are: full Jacobian generation using modern Forward Automated Differentiation (FAD) methods (Bartlett et al.,

2006), and MCSA as the inner linear solver. This method has several attractive properties. First, the first-order approximation to the Jacobian used in JFNK type methods is eliminated by generating the Jacobian explicitly with the model equations through FAD. Second, the Jacobian need not be explicitly formed by the user but is instead automated through FAD; this eliminates the complexity of hand-coding derivatives and has also been demonstrated to be more efficient computationally than evaluating difference derivatives. Third, unlike GMRES, MCSA does not build a subspace during iterations. Although the Jacobian must be explicitly formed to use MCSA, for problems that take more than a few GMRES iterations to converge the size of the Krylov subspace will grow beyond that of the Jacobian. Finally, using MCSA for the linear solve provides its benefits for preconditioning, potential resiliency, and parallelism.

To present these new developments this document is arranged in the following manner. First, in Chapter 2, the fundamentals of the Monte Carlo method for linear systems are presented. Using this background on Monte Carlo, synthetic acceleration methods are then presented and analyzed using a simple model transport problem. Next, in Chapter 3, the Monte Carlo methods developed in Chapter 2 are applied to the neutron transport problem. Specifically, the SP_N form of the Boltzmann neutron transport equation is developed and solved using the Monte Carlo methods. Using a difficult fission reactor fuel assembly criticality calculation to drive research and development for these Monte Carlo methods, several important issues regarding Monte Carlo solver applicability and performance will be discussed along with solutions to these problems. Using these solutions, the Monte Carlo methods are then verified against modern solution techniques for the SP_N equations in order verify correctness for the fuel assembly criticality calculation. In Chapter 4, the Monte Carlo methods are applied to the Navier-Stokes equations as a model nonlinear energy and momentum transport system. Additional difficulties that arise when using Monte Carlo methods in these types of systems will be presented along with the subsequently developed solutions. In Chapter 5, the Monte Carlo methods are parallelized using modern reactor physics techniques for particle transport with a full parallel scaling analysis provided using a leadership-class computing facility. Finally, the work is summarized in Chapter 6 and topics of future work derived from the results of this research presented.

Chapter 2

Monte Carlo Synthetic Acceleration Methods

An alternative approach to approximate matrix inversion is to employ Monte Carlo methods that sample a distribution with an expectation value equivalent to that of the inverted operator. Such methods have been in existence for decades with the earliest reference noted here a manuscript published in 1950 by Forsythe and Leibler (Forsythe and Leibler, 1950). In their outline, Forsythe and Leibler in fact credit the creation of this technique to J. Von Neumann and S.M. Ulam some years earlier than its publication. In 1952 Wasow provided a more formal explanation of Von Neumann and Ulam's method (Wasow, 1952) and Hammersley and Handscomb's 1964 monograph (Hammersley and Handscomb, 1964) and Spanier and Gelbard's 1969 book (Spanier and Gelbard, 1969) present additional detail on this topic using a collection of references from the 1950's and early 1960's.

Following this work, Halton presented a residual Monte Carlo scheme that improved dramatically on the performance of these methods (Halton, 1962). Recent work in this area has utilized these residual Monte Carlo methods in particle transport (Evans et al., 2003), giving accelerated convergence over traditional methods. As an expansion of Halton's work, Evans and others have leveraged these methods in a synthetic acceleration scheme that has been shown to be competitive with production Krylov methods for radiation transport problems and improve performance significantly over Halton's sequential method (Evans et al., 2012).

In this chapter, we will present the fundamentals of the Monte Carlo method for discrete linear systems. Both the forward and adjoint methods will be presented and analyzed including a brief variance analysis of several Monte Carlo estimators. The Sequential Monte Carlo method of Halton and Monte Carlo Synthetic Acceleration methods will then be presented as a means of leveraging the basic Monte Carlo methods devised by Neumann and Ulam in an iterative refinement scheme that accelerates their convergence. Using these iterative schemes, a set of modest variance reduction techniques will be introduced and their effects along with other parameters of the algorithms on the Monte Carlo solutions explored using a simple model transport problem.

2.1 Preliminaries

We seek solutions of the general linear problem in the following form:

$$\mathbf{A}\mathbf{x} = \mathbf{b}\,, \tag{2.1}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a matrix operator such that $\mathbf{A} : \mathbb{R}^N \to \mathbb{R}^N$, $\mathbf{x} \in \mathbb{R}^N$ is the solution vector, and $\mathbf{b} \in \mathbb{R}^N$ is the forcing term. The solutions to Eq (2.1) will be generated by inverting $\mathbf{A}$ either directly or indirectly:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}\,. \tag{2.2}$$

In addition we can define the residual:

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}\,, \tag{2.3}$$

such that an exact solution $\mathbf{x}$ has been found when $\mathbf{r} = \mathbf{0}$. From the statement in Eq (2.2) we can already place a restriction on $\mathbf{A}$ by requiring that it be *nonsingular*, meaning that we can in fact compute $\mathbf{A}^{-1}$. In this work we will focus our efforts on approximately inverting the operator through various means.

In a discussion of methods for solving linear systems, several mathematical tools are useful in characterizing the qualities of the linear system. Among the most useful are the *eigenvalues* of the matrix, $\sigma(\mathbf{A})$. We find these by solving the eigenvalue problem:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x},\ \lambda \in \sigma(\mathbf{A})\,. \tag{2.4}$$

By writing Eq (2.4) in a different form,

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0\,, \tag{2.5}$$

and demanding that non-trivial solutions for $\mathbf{x}$ exist, it is then required that $|\mathbf{A}-\lambda\mathbf{I}| = 0$. Expanding this determinant yields a characteristic polynomial in terms of λ with roots that form the set of eigenvalues, $\sigma(\mathbf{A})$. Each component of $\sigma(\mathbf{A})$ can then be used to solve Eq (2.5) for a particular permutation of $\mathbf{x}$. The set of all permutations form the *eigenvectors* of $\mathbf{A}$. A quantity of particular interest that is computable from the eigenvalues of a matrix $\mathbf{A}$ is the *spectral radius*, $\rho(\mathbf{A})$, defined by Saad (Saad, 2003) as:

$$\rho(\mathbf{A}) = \max_{\lambda\in\sigma(\mathbf{A})} |\lambda|\,. \tag{2.6}$$

2.2 Neumann-Ulam Method

We begin our discussion of Monte Carlo methods by seeking a solution to Eq (2.1). For a given linear operator $\mathbf{A}$, we can use diagonal splitting in a similar manner as the stationary method in Eq (A.3) to define the following operator[1]:

$$\mathbf{H} = \mathbf{I} - \mathbf{A} \,, \tag{2.7}$$

such that we are solving the system:

$$\mathbf{x} = \mathbf{H}\mathbf{x} + \mathbf{b} \,. \tag{2.8}$$

We can then form an alternative representation for $\mathbf{A}^{-1}$ by generating the *Neumann series*:

$$\mathbf{A}^{-1} = (\mathbf{I} - \mathbf{H})^{-1} = \sum_{k=0}^{\infty} \mathbf{H}^k \,, \tag{2.9}$$

which will converge if the spectral radius of $\mathbf{H}$ is less than 1. If we then apply this Neumann series to the right hand side of Eq (2.1) we acquire the solution to the linear problem:

$$\mathbf{A}^{-1}\mathbf{b} = \sum_{k=0}^{\infty} \mathbf{H}^k\mathbf{b} = \mathbf{x} \,. \tag{2.10}$$

An approximation of this summation by truncation will therefore lead to an approximation of the solution. If we expand the summation with a succession of matrix-vector multiply operations, we arrive at an alternative perspective of this summation by considering the i^{th} component of the solution vector:

$$x_i = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \dots \sum_{i_k}^{N} h_{i,i_1} h_{i_1,i_2} \dots h_{i_{k-1},i_k} b_{i_k} \,, \tag{2.11}$$

which can interpreted as a series of transitions between states,

$$\nu = i \to i_1 \to \cdots \to i_{k-1} \to i_k \,, \tag{2.12}$$

in $\mathbf{H}$ where ν is interpreted as a particular random walk sequence permutation. We can generate these sequences of transitions through Monte Carlo random walks by assigning them both a probability and weight. As a reinterpretation of the iteration

[1]It should be noted that non-diagonal splittings have been recently explored (Srinivasan, 2010) and have the potential to improve efficiency. However, it was observed in this work that this type of splitting did not improve performance in the asymptotic limit of $\rho(\mathbf{H}) \to 1$ for non-trivial problems.

matrix, we then form the *Neumann-Ulam decomposition* of $\mathbf{H}$:

$$\mathbf{H} = \mathbf{P} \circ \mathbf{W} \,, \tag{2.13}$$

where $\circ$ denotes the Hadamard product operation[2], $\mathbf{P}$ denotes the transition probability matrix, and $\mathbf{W}$ denotes the transition weight matrix. This decomposition, a generalization of Dimov's work (Dimov et al., 1998), is an extension of the original Neumann-Ulam scheme in that now a weight cutoff can be used to terminate a random walk sequence and therefore truncate the Neumann series it is approximating. The formulation of $\mathbf{P}$ and $\mathbf{W}$ will be dependent on whether we choose a direct or adjoint Monte Carlo sequence to estimate the state transitions in Eq (2.11). In the direct method, we will use the provided linear operator $\mathbf{A}$ to form the Neumann Ulam decomposition while the adjoint method will use the adjoint linear operator ($\mathbf{A}^T$ for real-valued systems) to form the decomposition.

2.3 Direct Monte Carlo Method

In the context of matrix inversion, a direct (forward) method resembles an adjoint Monte Carlo method in the reactor physics community where the solution state is sampled and the source terms that contribute to it are assembled. To achieve this, we build the forward Neumann-Ulam decomposition per Dimov's approach by first choosing a probability matrix that is a row scaling of $\mathbf{H}$ such that its components are:

$$p_{ij} = \frac{|h_{ij}|}{\sum_j |h_{ij}|} \,. \tag{2.14}$$

From this, we then see that the probability of transitioning from a state i to a state j is implicitly linked to the original operator $\mathbf{A}$ in that those terms with large values, and therefore those that make the greatest contribution to the numerical solution, will be sampled with a higher probability than smaller terms. In addition, the row scaling provides a normalization over the state to which we are transitioning such that $\sum_j p_{ij} = 1$, meaning that we sample the probabilities over the rows of the matrix. The components of the weight matrix are then defined by Eq (2.13) as:

$$w_{ij} = \frac{h_{ij}}{p_{ij}} \,. \tag{2.15}$$

[2]The Hadamard product $\mathbf{A} = \mathbf{B} \circ \mathbf{C}$ is defined element-wise as $a_{ij} = b_{ij}c_{ij}$.

It should be noted here that if $\mathbf{A}$ is sparse, then $\mathbf{H}$, $\mathbf{P}$, and $\mathbf{W}$ must be sparse as well by definition. Additionally, we only compute $\mathbf{P}$ and $\mathbf{W}$ from the non-zero elements of $\mathbf{H}$ as those components that are zero will not participate in the random walk and thus produce identical sparsity patterns for all matrices.

Using these matrices, we can then form the expectation value of the forward solution. For a given random walk permutation ν, we define the weight of that permutation on the m^{th} step to be:

$$W_m = w_{i_0,i_1} w_{i_1,i_2} \cdots w_{i_{m-1},i_m} \,, \tag{2.16}$$

such that the weight of each transition event contributes to the total through multiplication with $W_0 = 1$ as the starting weight. The contribution to the solution from a particular random walk permutation with k total events is then the *forward estimator*:

$$X_{i_0=i}(\nu) = \sum_{m=0}^{k} W_m b_{i_m} \,, \tag{2.17}$$

where $X_{i_0=i}(\nu)$ signifies that the solution state, i_0, in which the random walk ν started is also the state, i, in which we are tallying. We then define the probability that a particular random walk permutation of k events will occur:

$$P_\nu = p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{k-1},i_k} \,. \tag{2.18}$$

Finally, we define the expectation value of X to be the collection of all random walk permutations and their probabilities:

$$E\{X_i\} = \sum_{\nu} P_\nu X_i(\nu) \,, \tag{2.19}$$

which, if expanded, directly recovers the exact solution by forming the Neumann series:

$$\begin{aligned} E\{X_i\} &= \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \cdots \sum_{i_k}^{N} p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{k-1},i_k} w_{i,i_1} w_{i_1,i_2} \cdots w_{i_{k-1},i_k} b_{i_k} \\ &= x_i \,, \end{aligned} \tag{2.20}$$

therefore providing an unbiased Monte Carlo estimator.

In cases where we seek only approximate solutions, we need only to perform a predetermined number of random walks in order to generate an approximation for

x. If we are only to approximate the solution, we also need conditions by which we may terminate a random walk as the Neumann Ulam decomposition defined by Eqs (2.14) and (2.15) will create a random walk weight in Eq (2.16) that approaches, but never reaches zero. We do this by noticing that the factors added to Eq (2.16) will become diminishingly small due to their definition in Eq (2.15) and therefore their contributions to the solution estimate will become negligible. Using this, we choose terminate a random walk sequence with a *weight cutoff*, W_c, that is enforced when $W_m < W_c$ for a particular random walk permutation.

2.3.1 Forward Estimator Variance

We can compute the variance of the forward estimator through traditional methods by defining the variance, σ_i, for each component in the solution:

$$\sigma_i{}^2 = E\{X_i - (\mathbf{A}^{-1}\mathbf{b})_i\}^2 = E\{X_i^2\} - x_i^2 \,, \tag{2.21}$$

where the vector exponentials are computed element-wise. Inserting Eq (2.19) gives:

$$\sigma_i^2 = \sum_\nu P_\nu X_i(\nu)^2 - x_i^2 \,, \tag{2.22}$$

and applying Eq (2.17):

$$\sigma_i^2 = \sum_\nu P_\nu \Big(\sum_{m=0}^{k} W_m^2 b_{i_m}^2 + 2 \sum_{\substack{j=0 \\ m<j}}^{k} W_m W_j b_{i_m} b_{i_j} \Big) - x_i^2 \,. \tag{2.23}$$

We can distribute the random walk probability to yield an expanded form of the variance:

$$\sigma_i^2 = \sum_\nu P_\nu \sum_{m=0}^{k} W_m^2 b_{i_m}^2 + 2 \sum_\nu P_\nu \sum_{\substack{j=0 \\ m<j}}^{k} W_m W_j b_{i_m} b_{i_j} - x_i^2 \,. \tag{2.24}$$

In particular, we will isolate the first summation of the variance by expanding it:

$$\sum_\nu P_\nu \sum_{m=0}^{k} W_m^2 b_{i_m}^2 = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \cdots \sum_{i_k}^{N} p_{i,i_1} p_{i_1,i_2} \ldots p_{i_{k-1},i_k} w_{i,i_1}^2 w_{i_1,i_2}^2 \ldots w_{i_{k-1},i_k}^2 b_{i_k}^2 \,. \tag{2.25}$$

Using this expansion, we can arrive at a more natural reason for enforcing $\rho(\mathbf{H}) < 1$ for our Monte Carlo method to converge. Per the Hadamard product, we can concatenate

the summation in Eq (2.25):

$$(\mathbf{P} \circ \mathbf{W} \circ \mathbf{W})^k = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \cdots \sum_{i_k}^{N} p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{k-1},i_k} w_{i,i_1}^2 w_{i_1,i_2}^2 \cdots w_{i_{k-1},i_k}^2 \,. \tag{2.26}$$

If we assign $\mathbf{G} = \mathbf{P} \circ \mathbf{W} \circ \mathbf{W}$ as in Eq (2.13), we then have:

$$(\mathbf{P} \circ \mathbf{W} \circ \mathbf{W})^k = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \cdots \sum_{i_k}^{N} g_{i,i_1} g_{i_1,i_2} \cdots g_{i_{k-1},i_k} \,, \tag{2.27}$$

which is the general Neumann series for $\mathbf{G}$,

$$\mathbf{T} = \sum_{k=0}^{\infty} \mathbf{G}^k \,, \tag{2.28}$$

where $\mathbf{T} = (\mathbf{I} - \mathbf{G})^{-1}$. We can then insert T back into the variance formulation for a more concise definition:

$$\sigma_i^2 = (\mathbf{T}\mathbf{b}^2)_i + 2 \sum_{\nu} P_\nu \sum_{\substack{j=0 \\ m<j}}^{k} W_m W_j b_{i_m} b_{i_j} - x_i^2 \,. \tag{2.29}$$

We can relate $\mathbf{G}$ to $\mathbf{H}$ by noting that $\mathbf{G}$ simply contains an additional Hadamard product with the weight matrix. The Hadamard product has the property that:

$$|\mathbf{H} \circ \mathbf{W}| \geq |\mathbf{H}| \, |\mathbf{W}| \,. \tag{2.30}$$

Using the norm property of the Hadamard product and Eq (2.13), we can define the norm of $\mathbf{W}$ as:

$$\frac{|\mathbf{H}|}{|\mathbf{P}|} \geq |\mathbf{W}| \,. \tag{2.31}$$

Choosing the infinity norm of the operator, the row normalized probability matrix will yield a norm of 1 giving the following inequality for relating $\mathbf{G}$ and $\mathbf{H}$:

$$|\mathbf{G}| \geq |\mathbf{H}|^2 \tag{2.32}$$

Using these relations to analyze Eq (2.29), we see that if $\rho(\mathbf{G}) > 1$, then the first sum in Eq (2.28) will not converge and an infinite variance will arise as the elements of $\mathbf{T}$ become infinite in Eq (2.29). We must restrict $\mathbf{G}$ to alleviate this and therefore restrict $\mathbf{H}$ due to Eq (2.32) with $\rho(\mathbf{H}) < 1$ so that our expectation values for the solution may have a finite variance.

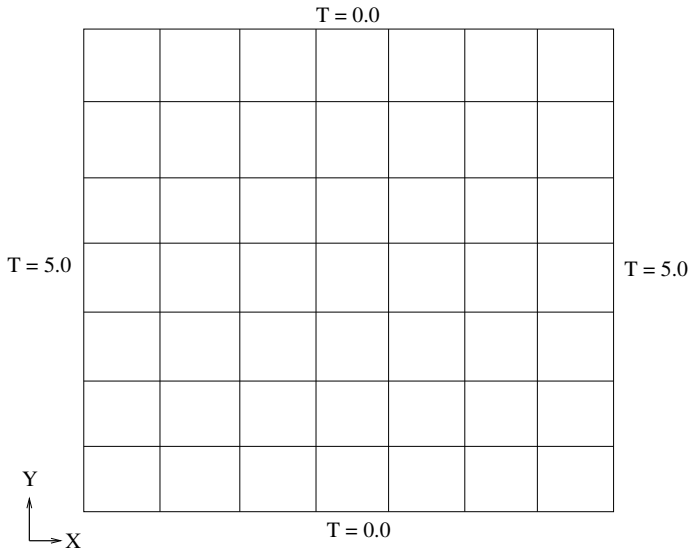

Figure 2.1: **Problem setup for 2D heat equation.** *Dirichlet conditions are set for the temperature on all 4 boundaries of the Cartesian grid. Background source of 1/5 the value of the boundary sources present.* 50×50 *grid.*

2.3.2 Direct Method: Evolution of a Solution

As a means of visually demonstrating the direct Monte Carlo method, consider a 2-dimensional thermal diffusion problem with sources on the left and right hand sides of the domain and a uniform source throughout the domain of 1/5 the strength of the boundary sources as shown in Figure 2.1. For this problem, the number of histories used to compute the solution at each grid point (state) in the domain was increased from 1 to 1000 in order to demonstrate the effects on the solution and the statistical nature of the method. Figure 2.2 gives these results. As the number of histories used per state is increased, the statistical variance of the solutions is decreased as more tallies are made. Starting with a single history at each state in the domain, the high variance prevents a precise solution from being obtained although we begin to see the solution take shape as expected. At 1000 histories per state, enough tallies have been made to generate a reasonable estimate for the structure of the solution. It is interesting to note here that as the statistical uncertainty is reduced at each grid point in the domain, the solution is resolved in a certain sense, analogous to the convergence of a traditional iterative method.

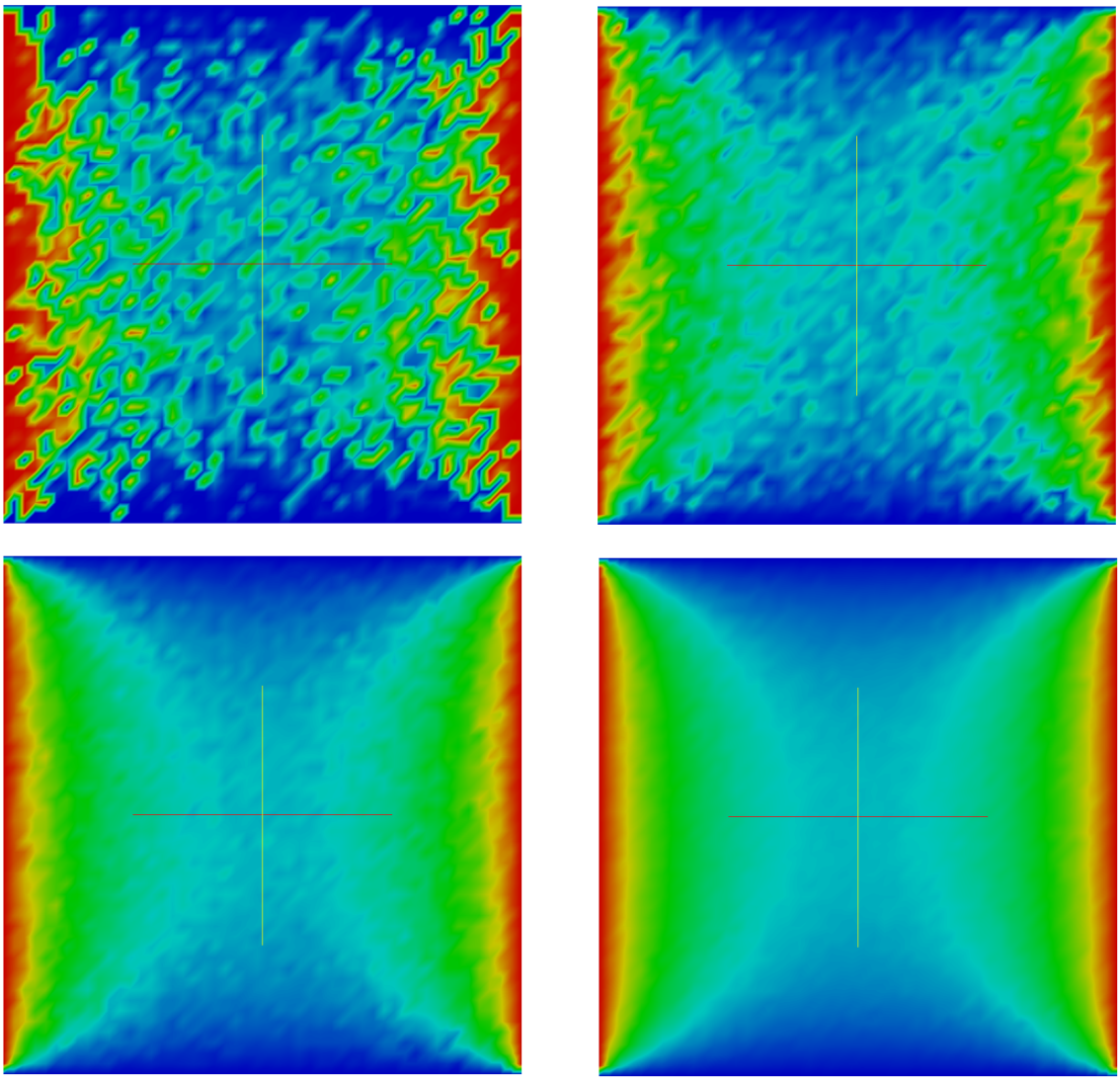

Figure 2.2: **Direct Monte Carlo solution to the heat equation with varying numbers of histories.** *Top left: 1 history per state. Top right: 10 histories per state. Bottom left: 100 histories per state. Bottom right: 1000 histories per state.*

2.4 Adjoint Monte Carlo Method

Often a more useful form, an alternative to forward Monte Carlo matrix inversion is the adjoint method. We begin by defining the linear system adjoint to Eq (2.1):

$$\mathbf{A}^T\mathbf{y} = \mathbf{d}\,, \tag{2.33}$$

where $\mathbf{y}$ and $\mathbf{d}$ are the adjoint solution and source vectors respectively and $\mathbf{A}^T$ is the adjoint operator for $\mathbf{A} \in \mathbb{R}^{N\times N}$. We can split this equation to mirror Eq (2.8):

$$\mathbf{y} = \mathbf{H}^T\mathbf{y} + \mathbf{d}\,. \tag{2.34}$$

As was required for convergence with the direct method using Eq (2.8), the spectral radius of $\mathbf{H}$ must remain less than 1 as $\mathbf{H}^T$ contains the same eigenvalues and therefore has the same spectral radius. By defining the following inner product equivalence (Spanier and Gelbard, 1969):

$$\langle \mathbf{A}^T\mathbf{x}, \mathbf{y}\rangle = \langle \mathbf{x}, \mathbf{A}\mathbf{y}\rangle\,. \tag{2.35}$$

it follows that:

$$\langle \mathbf{x}, \mathbf{d}\rangle = \langle \mathbf{y}, \mathbf{b}\rangle\,. \tag{2.36}$$

Using these definitions, we can derive an estimator from the adjoint method that will also give the solution vector, $\mathbf{x}$. As with the direct method, we can acquire the adjoint solution by forming the Neumann series by writing Eq (2.34) as:

$$\mathbf{y} = (\mathbf{I} - \mathbf{H}^T)^{-1}\mathbf{d}\,, \tag{2.37}$$

which in turn yields the Neumann series using the adjoint operator:

$$\mathbf{y} = \sum_{k=0}^{\infty}(\mathbf{H}^T)^k\mathbf{d}\,. \tag{2.38}$$

We expand this summation to again yield a series of transitions that can be approximated by a Monte Carlo random walk sequence, this time forming the Neumann series in reverse order:

$$y_i = \sum_{k=0}^{\infty}\sum_{i_1}^{N}\sum_{i_2}^{N}\ldots\sum_{i_k}^{N} h_{i_k,i_{k-1}}\ldots h_{i_2,i_1}h_{i_1,i}d_{i_k}\,. \tag{2.39}$$

We can readily build an estimator for the adjoint solution from this series expansion, but we instead desire the solution to Eq (2.1). Here we have 2 unknowns, $\mathbf{y}$ and $\mathbf{d}$, and therefore we require two constraints to close the system. We use Eq (2.36) as the first constraint and as a second constraint we select:

$$\mathbf{d} = \boldsymbol{\delta}_j \,, \tag{2.40}$$

where $\boldsymbol{\delta}_j$ is one of a set of vectors in which the j^{th} component is the Kronecker delta function $\delta_{i,j}$. If we apply Eq (2.40) to our first constraint Eq (2.36), we get the following convenient outcome:

$$\langle \mathbf{y}, \mathbf{b} \rangle = \langle \mathbf{x}, \boldsymbol{\delta}_j \rangle = x_j \,, \tag{2.41}$$

meaning that if we compute the inner product of the original source and the adjoint solution using a delta function source, we recover one component of the original solution.

In terms of radiation transport, this adjoint method is equivalent to a traditional forward method where the initial state i_0 of the random walk is determined by sampling the source vector $\mathbf{b}$ with probabilities:

$$P_{i_0=i}(\nu) = \frac{|b_i|}{||\mathbf{b}||_1} \,. \tag{2.42}$$

The random walk starting weight will then be $W_0 = b_{i_0}$. As a result of using the adjoint system, we modify our probabilities and weights using the *adjoint Neumann-Ulam decomposition* of $\mathbf{H}$:

$$\mathbf{H}^T = \mathbf{P} \circ \mathbf{W} \,, \tag{2.43}$$

where now we are forming the decomposition with respect to the transpose of $\mathbf{H}$. We then follow the same procedure as the direct method for forming the probability and weight matrices in the decomposition. Using the adjoint form, probabilities should instead be column-scaled:

$$p_{ij} = \frac{|h_{ji}|}{\sum_j |h_{ji}|} \,, \tag{2.44}$$

such that we expect to select a new state, j, from the current state in the random walk, i, by sampling column-wise (or row-wise if an adjoint probability matrix is formed).

Per Eq (2.43), the transition weight is then defined as:

$$w_{ij} = \frac{h_{ji}}{p_{ij}} \, . \tag{2.45}$$

Using the decomposition we can then define an expectation value for the adjoint method. Given Eq (2.16) as the weight generated for a particular random walk permutation as in Eq (2.12) and our result from Eq (2.41) generated by applying the adjoint constraints, the contribution to the solution in state j from a particular random walk permutation of k events is then the *collision estimator*:

$$X_j(\nu) = \sum_{m=0}^{k} W_m \delta_{i_m,j} \, , \tag{2.46}$$

where the Kronecker delta indicates that the tally contributes only in the current state, i_m, of the random walk. Note here that the estimator in Eq (2.46) does not have a dependency on the source state as in Eq (2.46), providing a remedy for the situation in the direct method where we must start a random walk in each source state for every permutation if we want to compute a solution estimate for that state. In the adjoint method, we instead tally in all states and those of lesser importance will not be visited as frequently by the random walk. Finally, the expectation value using all permutations is:

$$E\{X_j\} = \sum_{\nu} P_\nu X_j(\nu) \tag{2.47}$$

which, if expanded in the same way as the direct method, directly recovers the exact solution:

$$\begin{aligned} E\{X_j\} &= \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \dots \sum_{i_k}^{N} b_{i_0} h_{i_0,i_1} h_{i_1,i_2} \dots h_{i_{k-1},i_k} \delta_{i_k,j} \\ &= x_j \, , \end{aligned} \tag{2.48}$$

therefore also providing an unbiased Monte Carlo estimate of the solution. Note that this expansion produces the effective sequence of matrix-vector multiplications with the b_{i_0} component of the source vector which will be selected at random for each walk permutation and the unbiasedness of the estimator relies on an unbiased sampling of the source. It should also be noted here that Eq (2.48) only computes a single component of our desired solution vector when really what we desire is the entire solution vector. In an adjoint Monte Carlo simulation using this estimator, the w_{ij} elements that are added into the tally for each state are only selected if the random

walk currently resides in that state. Much like a mesh tally in a particle transport simulation, we have N simultaneous tallies for $\mathbf{A} \in \mathbb{R}^{N\times N}$ that will yield the entire solution vector. Based on their current state in the system, the random walks will contribute tally j in this group.

Like the direct method, we also desire a criteria for random walk termination for problems where only an approximate solution is necessary. For the adjoint method, we utilize a *relative weight cutoff*:

$$W_f = W_c b_{i_0} \,, \tag{2.49}$$

where W_c is defined as in the direct method. The adjoint random walk will then be terminated after m steps if $W_m < W_f$ as tally contributions become increasingly small.

2.4.1 Collision Estimator Variance

We can compute the variance of the collision estimator in the same way as the direct estimator for each component in the solution where now:

$$\sigma_j^2 = \sum_\nu P_\nu \sum_{m=0}^{k} W_m^2 \delta_{i_m,j} + 2\sum_\nu P_\nu \sum_{\substack{l=0 \\ m<l}}^{k} W_m W_l \delta_{i_m,j}\delta_{i_l,j} - x_j^2 \,. \tag{2.50}$$

In this case, in the second summation $m \neq l$ for all states due to the form of the summation and therefore the delta functions, $\delta_{i_m,j}$ or $\delta_{i_l,j}$, will only be nonzero for random walks that are in the current solution state (when $i_m = i_l = j$) and other states visited do not contribute to the variance of the current state (when $i_m \neq i_l$). This is important as it shows a decoupling of the variance among the solution vector states, meaning that the variance in solution state i does not depend on the variance in solution state j.

Expanding the transition probabilities in the first sum again yields a Neumann series in the same form as that explored for the forward estimator, this time with the terms in reverse order and the introduction of the Kronecker delta:

$$\sigma_j^2 = \sum_{k=0}^{\infty}\sum_{i_1}^{N}\sum_{i_2}^{N}\cdots\sum_{i_k}^{N} p_{i_k,i_{k-1}}\cdots p_{i_2,i_1}p_{i_1,i_0} w_{i_k,i_{k-1}}^2 \cdots w_{i_2,i_1}^2 w_{i_1,i_0}^2 b_{i_0}^2 \delta_{i_k,j} + \\ 2\sum_\nu P_\nu \sum_{\substack{l=0 \\ m<l}}^{k} W_m W_l \delta_{i_m,j}\delta_{i_l,j} - x_j^2 \,. \tag{2.51}$$

If we again define $\mathbf{G} = \mathbf{P} \circ \mathbf{W} \circ \mathbf{W}$, we then have:

$$\sigma_j^2 = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \cdots \sum_{i_k}^{N} g_{i_k,i_{k-1}} \cdots g_{i_2,i_1} g_{i_1,i_0} b_{i_0} \delta_{i_k,j} + 2 \sum_{\nu} P_\nu \sum_{\substack{l=0 \\ m<l}}^{k} W_m W_l \delta_{i_m,j} \delta_{i_l,j} - x_j^2 \,, \quad (2.52)$$

where now $\mathbf{G}$ in this case is the transpose of that used in Eq (2.27). The Kronecker delta again implies that the variance contribution from each random walk will only be in its current state and for many random walks, many starting weights of b_{i_0} will contribute to the variance in the same way as they contribute to the solution tally. As the transpose is formed and the Neumann series of $\mathbf{G}$ is in reverse order in Eq (2.52) relative to its formulation in the forward estimator, then Eq (2.52) and Eq (2.27) are equivalent and therefore the collision estimator is bound by the same restrictions on $\rho(\mathbf{G})$ and $\rho(\mathbf{H})$ to ensure that the variance is finite.

2.4.2 Expected Value Estimator

In addition to the collision estimator, an additional estimator is available due to the work of Okten (Okten, 2005) that uses the method of expected values as a means to improve the Monte Carlo estimate. As outlined by Spanier and Gelbard (Spanier and Gelbard, 1969), the method of expected values is a deterministic averaging of events that may potentially occur in the Monte Carlo random walk sequence. Okten applied this principle directly to discrete Monte Carlo by forming the *expected value estimator* for a random walk of k events:

$$X_j(\nu) = b_j + \sum_{m=0}^{k} W_m h_{j,i_m} \quad (2.53)$$

where now the contribution of the iteration matrix is deterministically averaged at step m over all potential states j that may be reached from the current state i_m. Via Okten, the estimator can be shown to be unbiased through a comparison to the expected value estimator. We can first rewrite the summation in Eq (2.53):

$$X_j(\nu) = b_j + \sum_{m=0}^{k} \sum_{i=1}^{N} W_m \delta_{i_m,i} h_{ji} \quad (2.54)$$

where N is the number of states in the system. Immediately, we see the collision estimator as defined by Eq (2.46) and can therefore write the expectation value as:

$$E\{X_j\} = b_j + \sum_{i=1}^{N} E\{X_i\} h_{ji} \tag{2.55}$$

which is equivalently is the j^{th} component of Eq (2.8):

$$E\{X_j\} = b_j + \sum_{i=1}^{N} x_i h_{ji} \tag{2.56}$$

and is therefore and unbiased estimate. Compared to the collision estimator, the expected value estimator provides additional information at every step of the random walk, yielding potentially better statistics with the same amount of transport work. Conveniently, even if no Monte Carlo histories are computed, the expected value estimator still deterministically computes the first term of the Neumann Series, $\mathbf{H}^0\mathbf{b}$, whereas the collision estimator will provide no information.

2.4.2.1 Expected Value Estimator Variance

Following the collision estimator variance, the expected value estimator variance is given as:

$$\sigma_j^2 = \sum_{\nu} P_\nu b_j^2 + 2 \sum_{\nu} P_\nu b_j \sum_{m=0}^{k} W_m h_{j,i_m} + \sum_{\nu} P_\nu \sum_{m=0}^{k} W_m^2 h_{j,i_m}^2 + 2 \sum_{\nu} P_\nu \sum_{\substack{l=0 \\ m<l}}^{k} W_m W_l h_{j,i_m} h_{j,i_l} - x_j^2 \,. \tag{2.57}$$

As the delta functions are no longer present, the final summation in Eq (2.57) contains contributions for all valid state combinations in the same row of the iteration matrix and therefore the variance of each state in the solution tally is coupled to a group of other states as defined by the sparsity pattern of the iteration matrix. This coupling of variances is expected due to the deterministic averaging used to generate the expected value estimator.

2.4.3 Adjoint Method: Evolution of a Solution

As a means of visually demonstrating the adjoint Monte Carlo method, again consider the 2-dimensional thermal diffusion problem with sources on the left and right hand

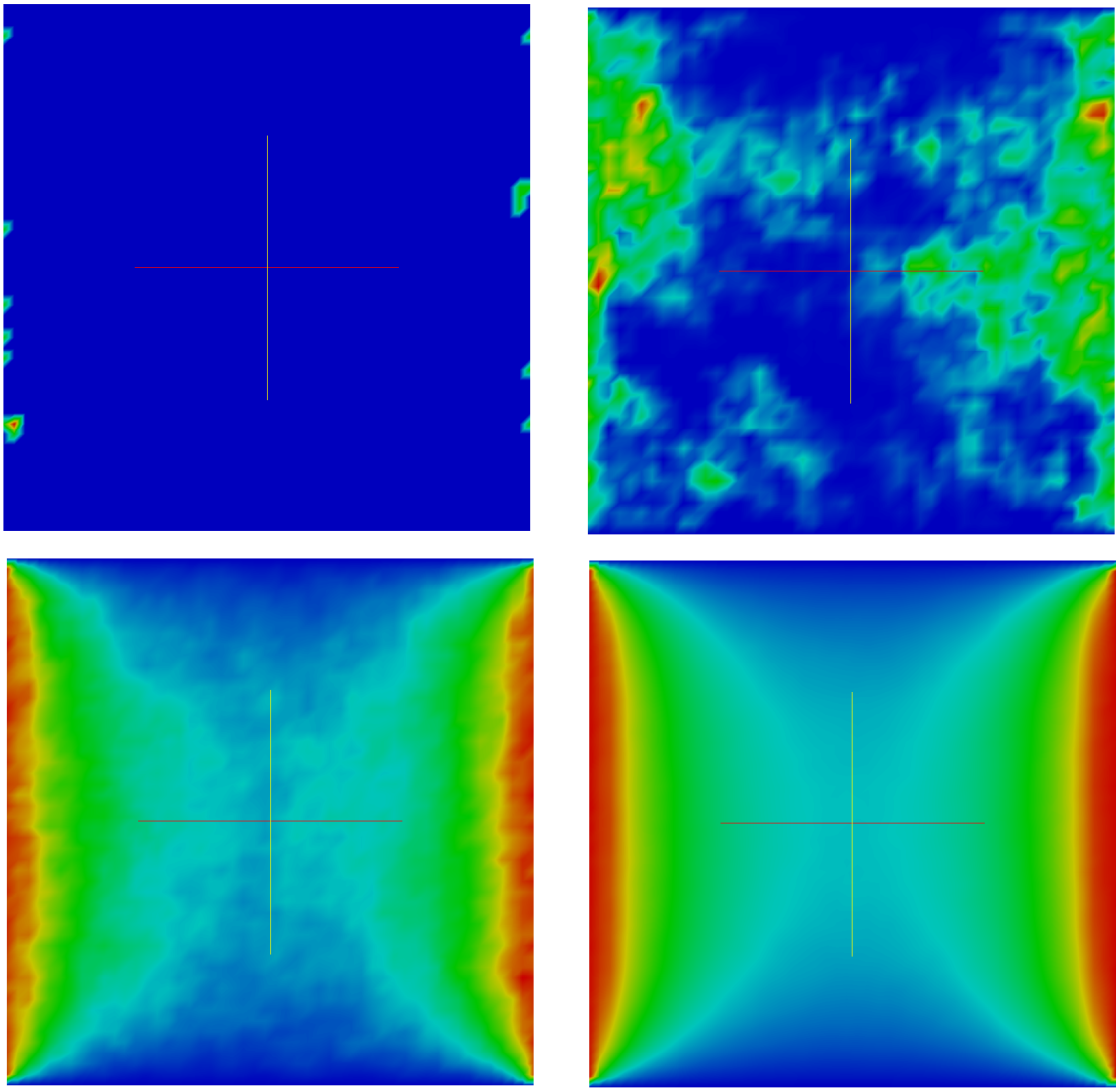

Figure 2.3: **Adjoint Monte Carlo solution to the heat equation with varying numbers of histories.** *Top left: 10 histories per state. Top right: 1,000 histories per state. Bottom left: 100,000 histories per state. Bottom right: 10,000,000 histories per state.*

sides of the domain and a smaller uniform source as shown in Figure 2.1. Using the adjoint method with the collision estimator, the number of histories sampled from the source was increased from 10 to 10,000,000 in order to show its effects on the solution and the statistical nature of the method. Figure 2.3 gives these results. As the number of histories used per state is increased, the statistical variance of the solutions is decreased as more tally contributions are made. At 10,000,000 histories per state, enough information has been tallied to generate a reasonable estimate for the structure

of the solution. The visual difference between Figures 2.2 and 2.3 is precisely that determined by their mathematics. As the adjoint solution evolves with the addition of histories, more histories emanate from the boundary the smaller uniform source in the domain with more penetrating from the boundary into the domain and making contributions to the tallies in those states as they are transported.

2.5 Sequential Monte Carlo

The direct and adjoint Neumann-Ulam methods described are limited by a convergence rate of $1/\sqrt{N}$ by the Central Limit Theorem where N is the number of random walk permutations. In 1962, Halton presented a residual Monte Carlo method that moves towards exponential convergence rates (Halton, 1962) and further refined his work some years later (Halton, 1994). Applications of his work by the transport community have confirmed convergence rates on the order of of e^{-N} (Evans et al., 2003). In much the same way as the projection methods outlined in Appendix A, Halton's method, sequential Monte Carlo, utilizes the adjoint Monte Carlo solver as a means of directly reducing the elements residual vector. He proposed the following iterative scheme as a solution to Eq (2.1)

$$\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k \,, \tag{2.58a}$$

$$\mathbf{A}\boldsymbol{\delta}^k = \mathbf{r}^k \,, \tag{2.58b}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \boldsymbol{\delta}^k \,, \tag{2.58c}$$

where the correction $\boldsymbol{\delta}^k$ is computed by the adjoint Monte Carlo method at each iteration. The merits of Halton's approach are immediately visible in that we have now broken the binding of the convergence rate to the Central Limit Theorem. Here, the Monte Carlo solver is used to produce a correction from the residual, analogous to using the residual to extract a correction from the search subspace in a projection method. By doing this, the Monte Carlo error is bound in the correction used to update the solution and therefore does not explicitly manifest itself in the overall convergence of the solution. The downside of such a method is that if the solution guess is poor, then many iterations are required in order to reach exponential converge as the Monte Carlo error (and therefore the Central Limit Theorem) does dominate in this situation.

2.6 Monte Carlo Synthetic Acceleration

Using the ideas of Halton, Evans and Mosher recently developed a Monte Carlo solution method that was not prohibited severely by the quality of the initial guess for the system (Evans and Mosher, 2009) and later applied it more rigorously as a solution mechanism for the radiation diffusion equation (Evans et al., 2012). With their new methods, they achieved identical numerical results as conventional Krylov solvers as well as comparable performance in both number of iterations and CPU time. Their approach was instead to use residual Monte Carlo as a synthetic acceleration for a stationary method. To derive this method, we begin by splitting the operator in Eq (2.1)

$$\mathbf{x} = (\mathbf{I} - \mathbf{A})\mathbf{x} + \mathbf{b} \,. \tag{2.59}$$

With this we can then define the stationary method *Richardson's iteration* as:

$$\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{A})\mathbf{x}^{k} + \mathbf{b} \,, \tag{2.60}$$

which will converge if $\rho(\mathbf{I} - \mathbf{A}) < 1$. We then define the solution error at the k^{th} iterate relative to the true solution:

$$\delta\mathbf{x}^{k} = \mathbf{x} - \mathbf{x}^{k} \,. \tag{2.61}$$

Subtracting Eq (2.60) from Eq (2.59) we get:

$$\delta\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{A})\delta\mathbf{x}^{k} \,. \tag{2.62}$$

Subtracting from this $(\mathbf{I} - \mathbf{A})\delta\mathbf{x}^{k+1}$ yields:

$$\begin{aligned} \mathbf{A}\delta\mathbf{x}^{k+1} &= (\mathbf{I} - \mathbf{A})(\mathbf{x}^{k+1} - \mathbf{x}^{k}) \\ &= \mathbf{r}^{k+1} \,. \end{aligned} \tag{2.63}$$

Using this, we define the following scheme that will converge in one iteration if $\mathbf{A}$ is inverted exactly:

$$\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{A})\mathbf{x}^{k} + \mathbf{b} \,, \tag{2.64a}$$

$$\mathbf{A}\delta\mathbf{x}^{k+1} = \mathbf{r}^{k+1} \,, \tag{2.64b}$$

$$\mathbf{x} = \mathbf{x}^{k+1} + \delta\mathbf{x}^{k+1} \,. \tag{2.64c}$$

However, $\mathbf{A}$ is only approximately inverted by our numerical methods and therefore we instead pose an iterative scheme in which the Monte Carlo solvers are used to invert the operator. The *Fixed-Point Monte Carlo Synthetic-Acceleration* (MCSA) method is defined as:

$$\mathbf{x}^{k+1/2} = \mathbf{x}^{k} + \mathbf{r}^{k} \,, \tag{2.65a}$$

$$\mathbf{r}^{k+1/2} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1/2} \,, \tag{2.65b}$$

$$\mathbf{A}\delta\mathbf{x}^{k+1/2} = \mathbf{r}^{k+1/2} \,, \tag{2.65c}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta\mathbf{x}^{k+1/2} \,, \tag{2.65d}$$

$$\mathbf{r}^{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1} \,, \tag{2.65e}$$

where a Neumann-Ulam Monte Carlo method is used to generate the solution correction from the residual and Richardson's iteration in the first step has been rewritten as a residual correction. Using Monte Carlo in this way achieves the same effect as Halton's method, decoupling its convergence rate from the overall convergence rate of the method. Here, the approximate Monte Carlo solution is not driven to a particular convergence as it merely supplies a correction for the initial guess generated by Richardson's iteration. Rather, only a set number of histories are required using the Neumann-Ulam method to generate the correction. In addition, the fact that the scheme in Eq (2.64) will converge in one iteration if $\mathbf{A}$ is inverted exactly means that as more and more stochastic histories are used to compute the correction and the error is reduced towards zero, the number of iterations required for MCSA to converge should decrease accordingly, thus accelerating the solution.

In addition to the Monte Carlo solver parameters dictating the number of histories and weight cutoff, the outer MCSA iterations also have the following stopping criteria:

$$||\mathbf{r}||_{\infty} < \epsilon \, ||\mathbf{b}||_{\infty} \,, \tag{2.66}$$

where ϵ is a user-defined parameter. As with any iterative method, other stopping criteria using other vector norms could be computed, however, for this work we will only use Eq (2.66). We therefore have 3 parameters to tune in an MCSA implementation: the number of Monte Carlo histories computed in the Neumann-Ulam solve during each MCSA iteration, the weight cutoff for those histories, and the total MCSA convergence tolerance as specified by ϵ.

2.6.1 Alternative Fixed Point Iterations

In addition to the basic Richardson iteration, the MCSA algorithm presented in Eq (2.65) can be used to accelerate any fixed point iteration which only depends on the state (typically the residual) of the last iteration. As outlined in Appendix A, subspace methods take on a general form using the Petrov-Galerkin conditions as constraints for extracting a correction from the search subspace as given by Eq (A.16). Those that are one-dimensional may be used with fixed-point MCSA as they only depend the previous iteration for information. As an example, for positive-definite but not necessarily symmetric problems, the minimal residual (MINRES) iteration (Saad, 2003) can be used in conjunction with MCSA where the residual vector, $\mathbf{r}$, defines the search subspace and the action of the linear operator on the residual vector, $\mathbf{Ar}$, defines the constraint subspace. Using this, we can then define an MCSA scheme that accelerates the MINRES iteration:

$$\alpha = \frac{\langle \mathbf{A}\mathbf{r}^k, \mathbf{r}^k \rangle}{\langle \mathbf{A}\mathbf{r}^k, \mathbf{A}\mathbf{r}^k \rangle} , \tag{2.67a}$$

$$\mathbf{x}^{k+1/2} = \mathbf{x}^k + \alpha \mathbf{r}^k , \tag{2.67b}$$

$$\mathbf{r}^{k+1/2} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1/2} , \tag{2.67c}$$

$$\mathbf{A}\delta\mathbf{x}^{k+1/2} = \mathbf{r}^{k+1/2} , \tag{2.67d}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta\mathbf{x}^{k+1/2} , \tag{2.67e}$$

$$\mathbf{r}^{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1} . \tag{2.67f}$$

where α is an optimal extrapolation parameter generated from the constraints. Interestingly, this scheme is nearly identical to the Richardson iteration version in Eq (2.65) except now the additional extrapolation parameter, α, is applied to the residual correction in Eq (2.67b) as a means of selecting a more optimal search direction at each iteration, potentially further accelerating convergence.

2.6.2 Preconditioning MCSA

In most cases, at least a minimal amount of *preconditioning* of the linear system will be required in order to use the class of stochastic methods described. Although these methods have no symmetry requirements for convergence, they do require that the spectral radius of the iteration matrix be less than one. Preconditioning serves as a means of achieving this by altering the eigenvalue spectrum of the iteration matrix.

2.6.2.1 Basic Preconditioning

As an example of basic preconditioning, to achieve a spectral radius of less than one for diagonally dominant matrices point Jacobi preconditioning can be used such that the preconditioning matrix $\mathbf{M}$ is:

$$\mathbf{M} = diag(\mathbf{A}) \,, \tag{2.68}$$

which may be trivially inverted. With the application of this preconditioner we are instead solving the following scaled linear system:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \,. \tag{2.69}$$

Next, we can apply MCSA to solve Eq (2.69):

$$\mathbf{x}^{k+1/2} = \mathbf{x}^{k} + \mathbf{M}^{-1}\mathbf{r}^{k} \,, \tag{2.70a}$$

$$\mathbf{r}^{k+1/2} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1/2} \,, \tag{2.70b}$$

$$\mathbf{M}^{-1}\mathbf{A}\delta\mathbf{x}^{k+1/2} = \mathbf{M}^{-1}\mathbf{r}^{k+1/2} \,, \tag{2.70c}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta\mathbf{x}^{k+1/2} \,, \tag{2.70d}$$

$$\mathbf{r}^{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1} \,, \tag{2.70e}$$

where the Neumann-Ulam Monte Carlo solve now has a preconditioned operator from which to build weights and probabilities for transport and a preconditioned source vector to sample.

Choosing point Jacobi preconditioning with MCSA is advantageous for several reasons. First, $\rho(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}) < 1$ is true for all $\mathbf{A}$ that is diagonally dominant and is easy to formulate because the inversion of $\mathbf{M}$ is trivial. Second, because the Monte Carlo method used within MCSA to compute the correction operates on a linear problem with the preconditioned operator, then $\mathbf{H}$ in the Neumann-Ulam solver will have a zero term in each of its diagonal elements, thereby eliminating all in-state transitions during the random walk sequence. Because of this, point Jacobi preconditioning should be considered for many classes of problems, regardless of any other preconditioning that is applied to the system.

2.6.2.2 General Preconditioning Strategies

It is possible to use general left, right, and left/right preconditioning with MCSA by carefully considering the underlying Monte Carlo problem that will be solved with the Neumann-Ulam method. We consider here the general left/right preconditioned method as the left or right preconditioned methods can be inferred from its formulation. We consider a left preconditioner $\mathbf{M_L}$ and a right preconditioner $\mathbf{M_R}$. The left/right preconditioned linear problem is then:

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\mathbf{M}_R\mathbf{x} = \mathbf{M}_L^{-1}\mathbf{b}\,. \tag{2.71}$$

To handle the right preconditioning, the system is written with a substitution of variables:

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\mathbf{u} = \mathbf{M}_L^{-1}\mathbf{b}\,, \tag{2.72}$$

with

$$\mathbf{x} = \mathbf{M}_R^{-1}\mathbf{u}\,. \tag{2.73}$$

To apply such a method to MCSA, we solve for the substituted variable $\mathbf{u}$ during the iteration sequence:

$$\mathbf{u}^{k+1/2} = \mathbf{u}^k + \mathbf{r}^k\,, \tag{2.74a}$$

$$\mathbf{r}^{k+1/2} = \mathbf{M}_L^{-1}(\mathbf{b} - \mathbf{A}\mathbf{M}_R^{-1}\mathbf{u}^{k+1/2})\,, \tag{2.74b}$$

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\delta\mathbf{u}^{k+1/2} = \mathbf{r}^{k+1/2}\,, \tag{2.74c}$$

$$\mathbf{u}^{k+1} = \mathbf{u}^{k+1/2} + \delta\mathbf{u}^{k+1/2}\,, \tag{2.74d}$$

$$\mathbf{r}^{k+1} = \mathbf{M}_L^{-1}(\mathbf{b} - \mathbf{A}\mathbf{M}_R^{-1}\mathbf{u}^{k+1})\,, \tag{2.74e}$$

and then recover the original solution vector with Eq (2.73). For the Monte Carlo problem, we isolate the generation of the correction:

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\delta\mathbf{u}^{k+1/2} = \mathbf{r}^{k+1/2}\,, \tag{2.75}$$

and note that the preconditioned residual of the substituted variable is now serving as the source and the new iteration matrix is:

$$\mathbf{H} = \mathbf{I} - \mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\,. \tag{2.76}$$

As we require (i, j) element-wise access to the iteration matrix in order to construct probabilities and weights for the Monte Carlo procedure from the Neumann-Ulam decomposition, the *composite operator*, $\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}$, must be formed via matrix-matrix multiplication.

Several possible shortcomings of this preconditioning approach are readily observed. First, the matrix-matrix multiplication operation for sparse, parallel distributed matrices is significantly more expensive than a matrix-vector multiplication operation. Second, each preconditioner must be explicitly inverted, an operation in itself that may be expensive and which prohibits the use of any preconditioners which provide no mechanism to extract their inverse. Third, for many modern preconditioning methods, this inversion may yield dense matrices, destroying sparsity and further impeding the performance of a matrix-matrix multiplication operation. It is also interesting to note that the Monte Carlo problem in the general left/right preconditioned scheme given by Eq (2.75) is not fully left/right preconditioned (meaning that we do not recover $\mathbf{x}$), but instead part of a sequence for finding the substituted variable $\mathbf{u}$. We do, however, gain the benefits of this general preconditioning by building the iteration matrix in Eq (2.76) from the fully preconditioned linear operator. In addition, for MCSA to apply to increasingly difficult problems, more advanced preconditioning techniques that require the generation of the composite operator may be necessary for convergence.

2.7 Monte Carlo Method Selection

The MCSA method defined in Eq. (2.65) uses the adjoint method to estimate the error in a residual Monte Carlo solve instead of the direct method outlined in §2.3. To demonstrate the effectiveness of the adjoint method over the direct method within the context of MCSA, we choose the two-dimensional time-dependent Poisson equation as a simple model transport problem[3]:

$$\frac{\partial \mathbf{u}}{\partial t} = \nabla^2 \mathbf{u} \,. \tag{2.77}$$

For all comparisons, a single time step is computed with backwards Euler time integration. The Laplacian is differenced on a square Cartesian grid with a second-

[3]The Poisson equation is simple form of the diffusion equation and has a similar elliptic character in its time-dependent form to the neutron transport equations that will be solved in the next chapter.

order five-point stencil,

$$\nabla_5^2 = \frac{1}{\Delta^2}[u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}] \,, \tag{2.78}$$

and a fourth-order nine-point stencil,

$$\nabla_9^2 = \frac{1}{6\Delta^2}[4u_{i-1,j} + 4u_{i+1,j} + 4u_{i,j-1} + 4u_{i,j+1} + u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} - 20u_{i,j}] \,, \tag{2.79}$$

both assuming a grid size of Δ in both the i and j directions. For a single time step solution, we then have the following sparse linear system to be solved with the MCSA method:

$$\mathbf{A}\mathbf{u}^{n+1} = \mathbf{u}^n \,. \tag{2.80}$$

Both the stencils will be used to vary the size and density of the sparse linear system in Eq. (2.80).

A timing and convergence study is used to demonstrate the effectiveness of the adjoint method with the collision estimator as compared to the direct method. To assess both the CPU time and number of iterations required to converge to a solution, a problem of constant Δ was used with varying values of the number of mesh elements, fixing the spectral radius of the system at a constant value for each variation. Both the five-point and nine-point stencils were used with both the direct and adjoint solvers. For each case, $N \times N$ total random walk permutations were computed per MCSA iteration where $N \times N$ is the number of discrete grid points in the system. Solver parameters were set to a weight cutoff of 1×10^{-4} for the stochastic linear solver and a convergence tolerance of 1×10^{-8} for the MCSA iterative solver. Figure 2.4 gives the CPU time needed for each case to converge in seconds and Figure 2.5 gives the number of iterations needed for each case to converge to the specified tolerance as a function of the problem size. All computations presented in this section and the remaining sections of this chapter were completed on a 3.0 GHz Intel Core 2 Quad Q9650 CPU machine with 16 GB 1067 MHz DDR3 memory.

We see clearly in Figure 2.4 that the using the adjoint solver with MCSA results in a speedup over the direct solver while the number of iterations required to converge is also reduced as shown in Figure 2.5. We expect this for several reasons. First, with an equivalent number of histories specified for both solvers per MCSA iteration and a system of size $N \times N$, the direct solver will compute a single random walk for each state in the system per iteration to acquire a solution in that state, regardless of the

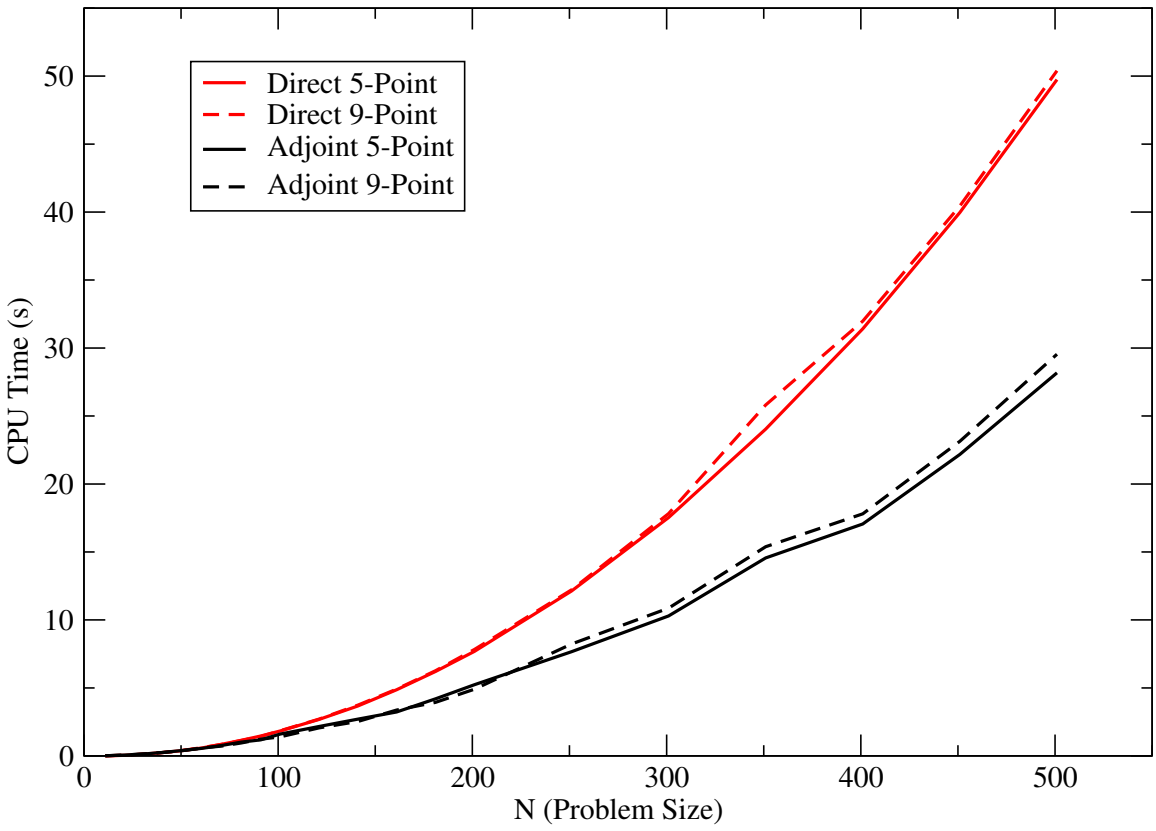

Figure 2.4: **CPU Time (s) to converge vs. Problem Size (N for an $N \times N$ square mesh).** *Both the adjoint and direct solvers are used with the five point and nine point stencils. A CPU time speedup is noted with the adjoint method due to the higher density of random walk events in regions with a large residual.*

size of the residual in that state. This is necessary in the direct method to ensure a contribution from each state as the random walk sequence will only contribute to the starting state. For the adjoint method, a total of $N \times N$ random walk events will have their starting state determined by sampling the residual vector. Because the random walk sequence contributes to the state in which it currently resides, sampling the residual vector as the Monte Carlo source gives a higher density of random walk events in regions with a high residual, thus giving a more accurate correction in that region due to reduced statistical error. From an iteration perspective, Figure 2.5 shows that using the direct method yields a roughly unchanging number of iterations required to converge as the problem size increases. Again, if we desire a correction value for all states in the problem, then we must start a random walk in each state in the system which does not reduce the number of iterations need as the problem size grows. Conversely, as the problem size grows in the adjoint method, the additional

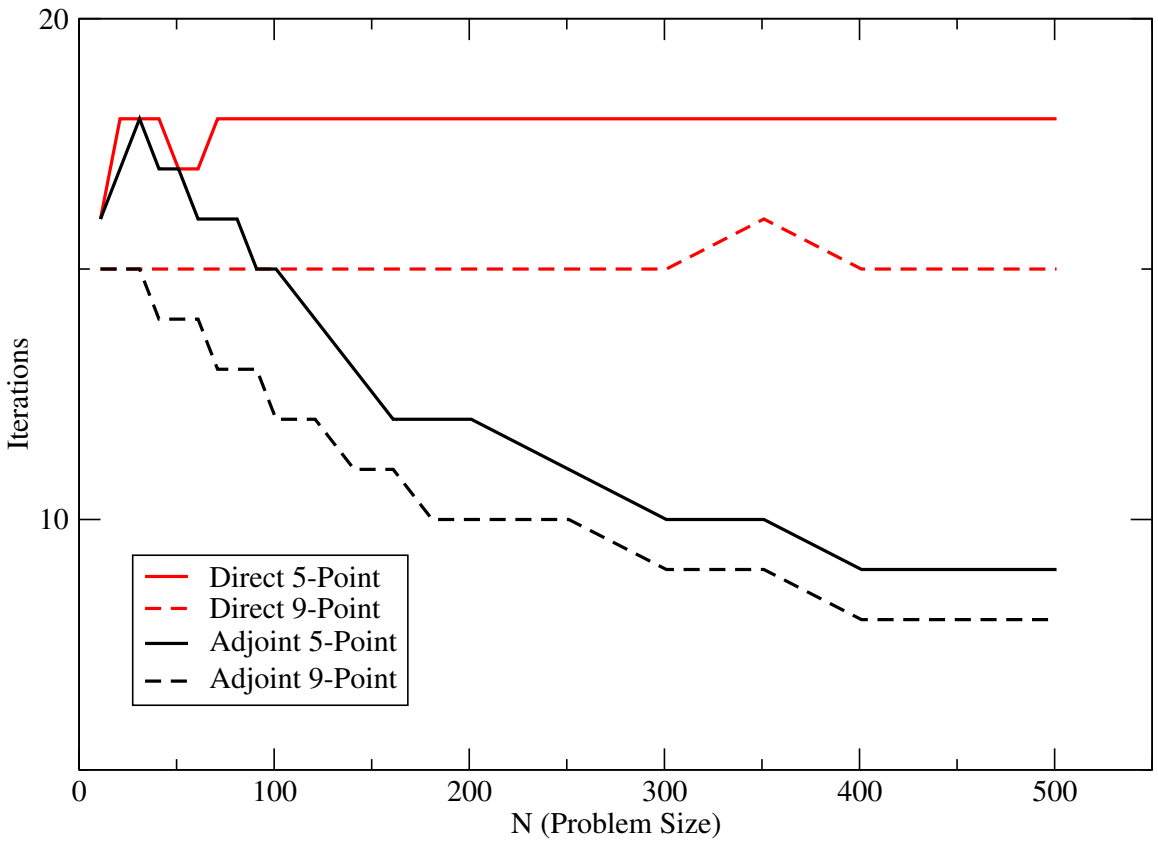

Figure 2.5: **Iterations to converge vs. Problem Size (N for an $N \times N$ square mesh).** *Both the adjoint and direct solvers are used with the five-point and nine-point stencils.*

stochastic histories that will be computed are concentrated in regions with a large residual, further reducing the stochastic error in the correction in those regions and subsequently reducing the required number of iterations to converge.

As an additional comparison, the convergence behavior of MCSA can be analyzed using both the adjoint and direct solvers to detect any performance benefits. To assess the convergence properties of MCSA using each solver and stencil, the infinity norm of the residual computed in Eq. (2.65) was collected at each iteration for a fixed problem size of $N = 500$. Figure 2.6 gives the results of these computations. First, it is worthy to note on the semi-log plot that we are indeed achieving the expected exponential convergence from MCSA with both Monte Carlo solvers. Second, we note that using the adjoint method with the same number of stochastic histories per MCSA iteration gives a faster rate of converge for the same reasons as above. We also note here that fewer iterations are required for convergence when the 9-point stencil

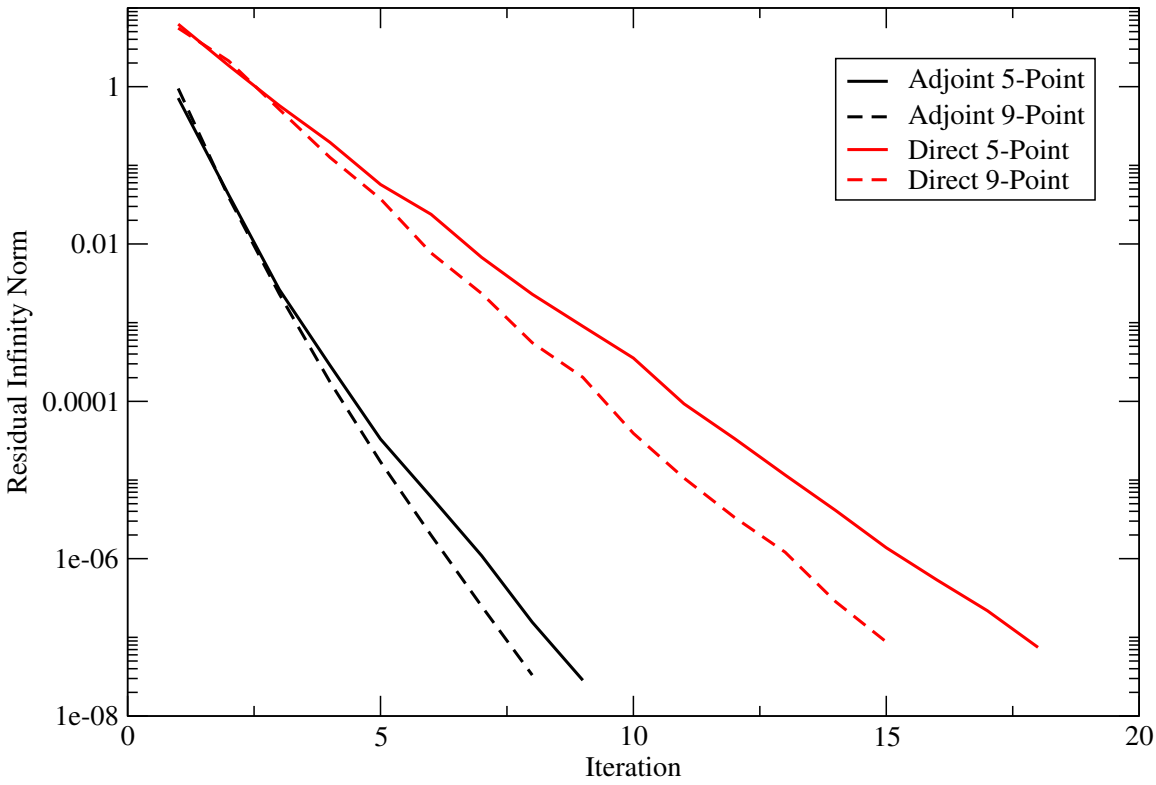

Figure 2.6: **Infinity norm of the solution residual vs. iteration number for a problem of size** $N = 500$**.** *Both the adjoint and direct solvers are used with the five point and nine point stencils. A higher rate of convergence is observed for MCSA using the adjoint Monte Carlo solver as compared to the direct method when both solvers compute the same number of random walks per iteration.*

is used to discretize the Laplacian operator (although at no gain in speed as given by the results in Figure 2.4). This is due to the fact that the smaller discretization error directly corresponds to a more well defined residual source generated by the Richardson extrapolation for the Monte Carlo calculation. In addition, the better defined source is transported through a domain described more accurately by the 9-point stencil, thus yielding a more accurate correction vector from the Monte Carlo calculation.

2.8 MCSA Comparison to Sequential Monte Carlo

To further motivate using Monte Carlo Synthetic Acceleration, we compare its performance to Halton's Sequential Monte Carlo method on which previous work in this area was based. For this comparison, we use the same transient Poisson problem as described in the previous section and choose only the 5-point stencil to discretize the Laplacian operator as the previous results yielded little qualitative difference between the discretizations. Both MCSA and Halton's method are used with the adjoint Monte Carlo solver and the collision estimator. In order to complete the same study as in the previous section, the number of histories computed by the Monte Carlo solver at each iteration had to be doubled to $2 \times N \times N$ in order to ensure convergence in Sequential Monte Carlo Method. For the majority of the problems in the previous section, the Sequential method used with $N \times N$ histories would not converge. Figure 2.7 gives the CPU time results for this comparison as a function of problem size while Figure 2.8 gives the number of iterations to converge as a function of problem size with a convergence tolerance of 1×10^{-8}. In both cases, using the Monte Carlo solver as a synthetic acceleration rather than in a pure residual Monte Carlo scheme resulted in a reduction in both CPU time and iterations required to converge. The additional Richardson extrapolation between each Monte Carlo solve in the MCSA method gives a better converged residual source to use with the Monte Carlo calculation while the Sequential method requires more iterations to achieve the same level of convergence in the residual.

The benefits of using a synthetic acceleration scheme are also noted when the infinity norm of the residual computed at each iteration for both methods was collected at each iteration for a fixed problem sizes of $N = 100$ and $N = 500$ as shown in figures Figure 2.9 and 2.10 respectively. In both cases, the Sequential method is subject to two regimes of exponential convergence with high frequency error modes removed in the first regime leaving lower frequency and slower converging error modes in the second. Using MCSA we a see a single rate of exponential convergence observed to be much higher than that computed by Halton's method due to the fact that the extra Richardson iteration is providing a smoothing effect to alleviate the error mode variations. Even with the doubling of the number of stochastic histories computed per time step in order to ensure convergence for the Sequential method, we still see robustness issues with a non-monotonically decreasing residual observed for the

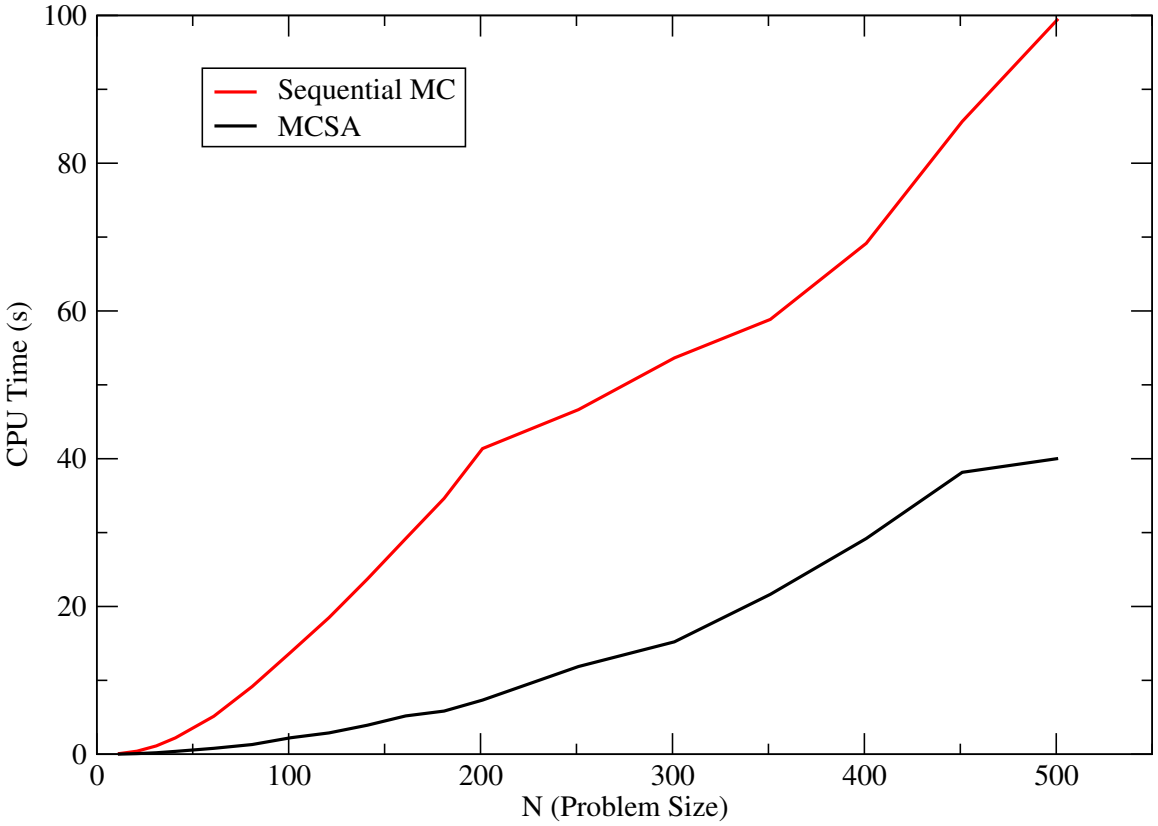

Figure 2.7: **CPU Time (s) to converge vs. Problem Size (N for an $N \times N$ square mesh).** *Both the Sequential Monte Carlo and MCSA solvers are used with the five point stencils and the adjoint Monte Carlo solver. The number of random walks was twice the number of discrete states in the system in order to ensure convergence in the Sequential Monte Carlo method.*

$N = 100$ case. In both cases the MCSA solver is observed to be robust with a monotonically decreasing residual.

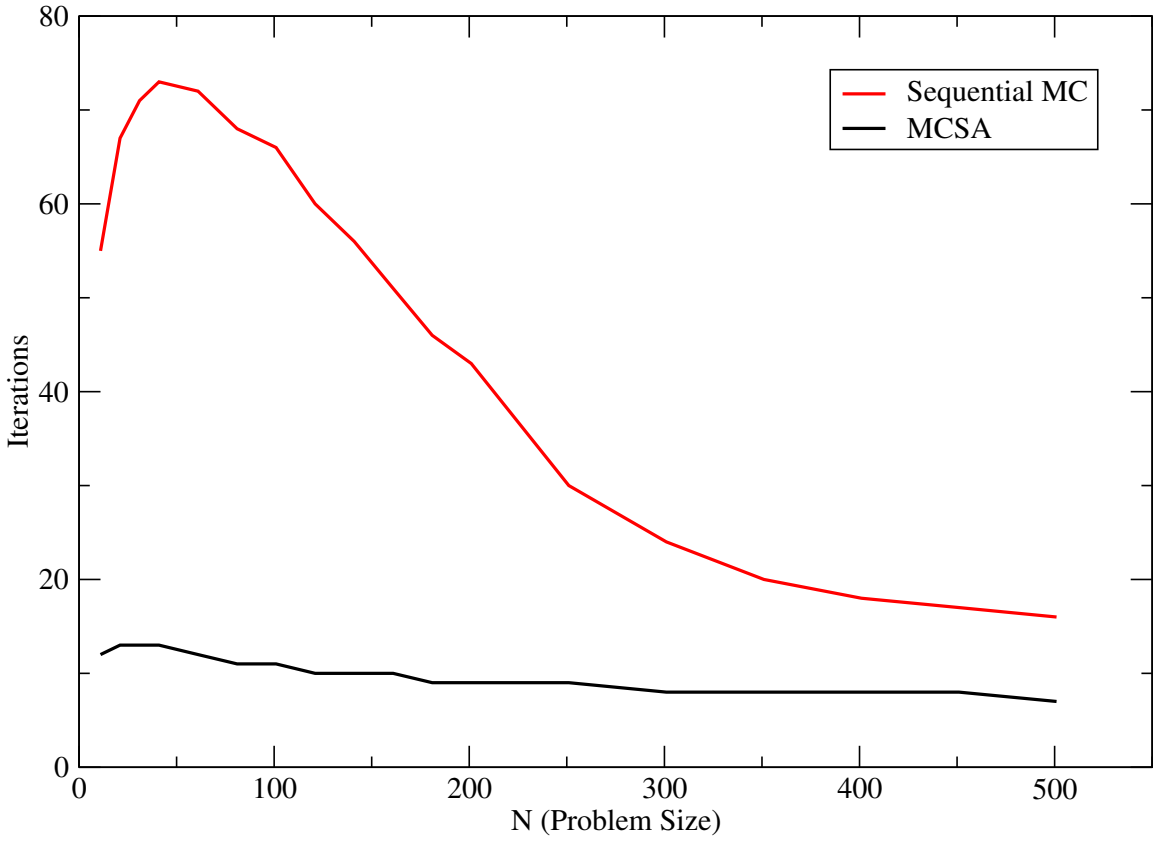

Figure 2.8: **Iterations to converge vs. Problem Size (N for an $N \times N$ square mesh).** *Both the Sequential Monte Carlo and MCSA solvers are used with the five point stencils and the adjoint Monte Carlo solver.*

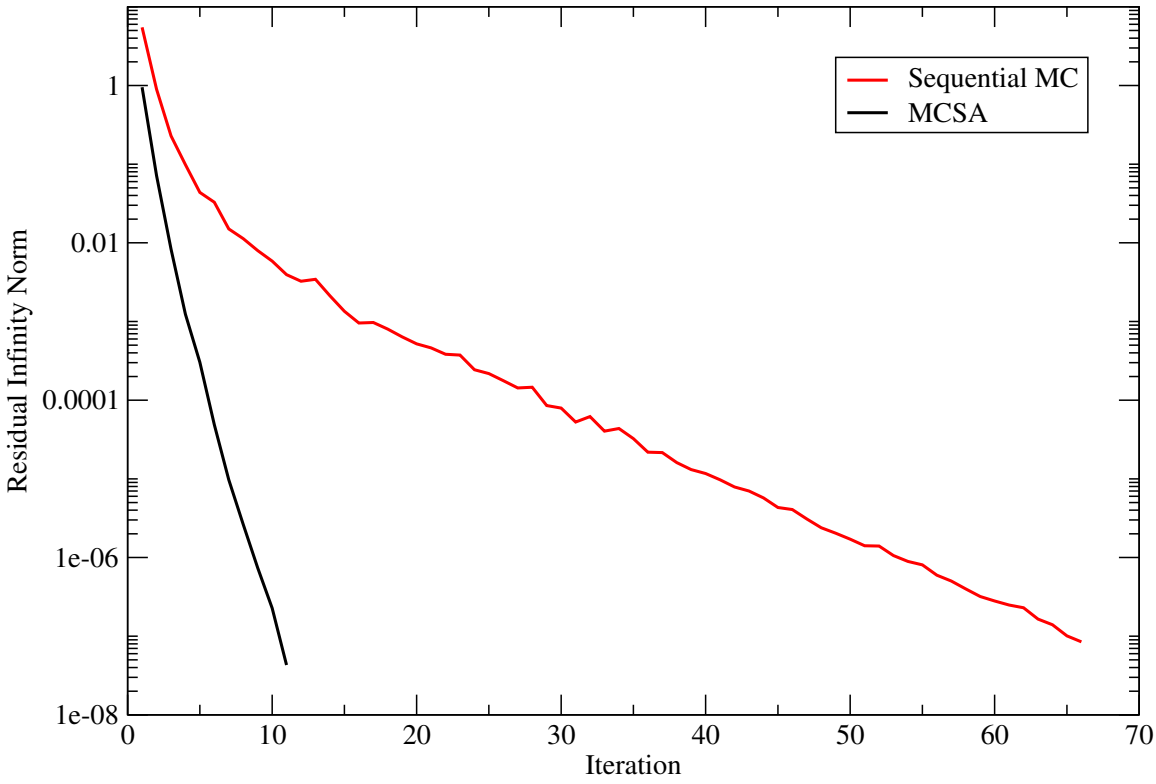

Figure 2.9: **Infinity norm of the solution residual vs. iteration number for a problem of size** $N = 100$**.** *Both the Sequential Monte Carlo and MCSA solvers are used with the five point stencils and the adjoint Monte Carlo solver.*

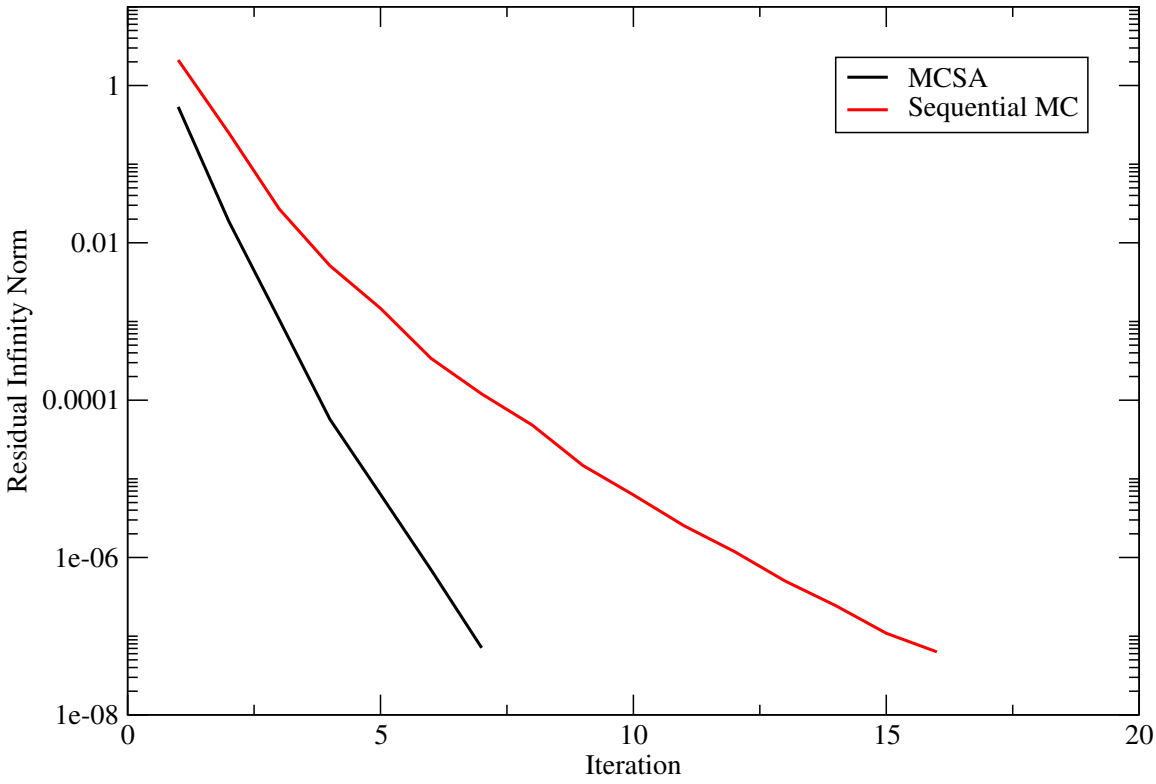

Figure 2.10: **Infinity norm of the solution residual vs. iteration number for a problem of size** $N = 500$**.** *Both the Sequential Monte Carlo and MCSA solvers are used with the five point stencils and the adjoint Monte Carlo solver.*

2.9 Monte Carlo Parameter and Estimator Analysis

With the adjoint method shown to be more effective than the direct method and MCSA to have better iterative and timing performance than sequential Monte Carlo, we now aim to study the effects of the adjoint Monte Carlo parameters, weight cutoff and number of histories, on MCSA performance with both the collision and expected value estimators. With the same Poisson problem, we will use a 200×200 grid and a convergence tolerance of 1×10^{-8} for all calculations. To study the effects of the number of Monte Carlo histories per MCSA iteration, the weight cutoff was fixed at 1×10^{-4} while the number of histories per iteration was varied from 6,000 to 100,000. It was observed that MCSA would not converge for this problem using the collision estimator with less than 6,000 histories while the expected value estimator permitted convergence with only 2,000 histories.

Figure 2.11 gives the number of iterations required to converge for both estimators as a function of the number of histories per iteration. For smaller numbers of histories, the performance of the expected value estimator is significantly better than the collision estimator. This result is valuable in that less transport is required to achieve the same MCSA iterative performance with the expected value estimator, important for situations where transport is expensive (i.e. in domain decomposed calculations). Interestingly, as the number of histories per iteration are increased, the iterative performance with the collision estimator approaches that of the expected value estimator. However, given the performance of the expected value estimator at a fractional number of histories, one should strongly consider its use over using the collision estimator with more histories. The CPU time required to converge is presented in Figure 2.12 and reflects the results of the iterative performance. In general, using the collision estimator is slightly slower overall, but the time per iteration is faster due to the fact that the estimator does not have to cycle through multiple states during the tally procedure.

For the weight cutoff study, the number of histories per iteration was fixed at 40,000 and the weight cutoff varied from 5×10^{-1} down to 1×10^{-10}. Surprisingly, the number of iterations to converge given by Figure 2.13 is effectively invariant to the weight cutoff, only seeing detrimental effects on the number of iterations at a very large weight cutoff. This suggests that a fairly large weight cutoff can be used in practice with both estimators and that the preliminary components of the random walk are

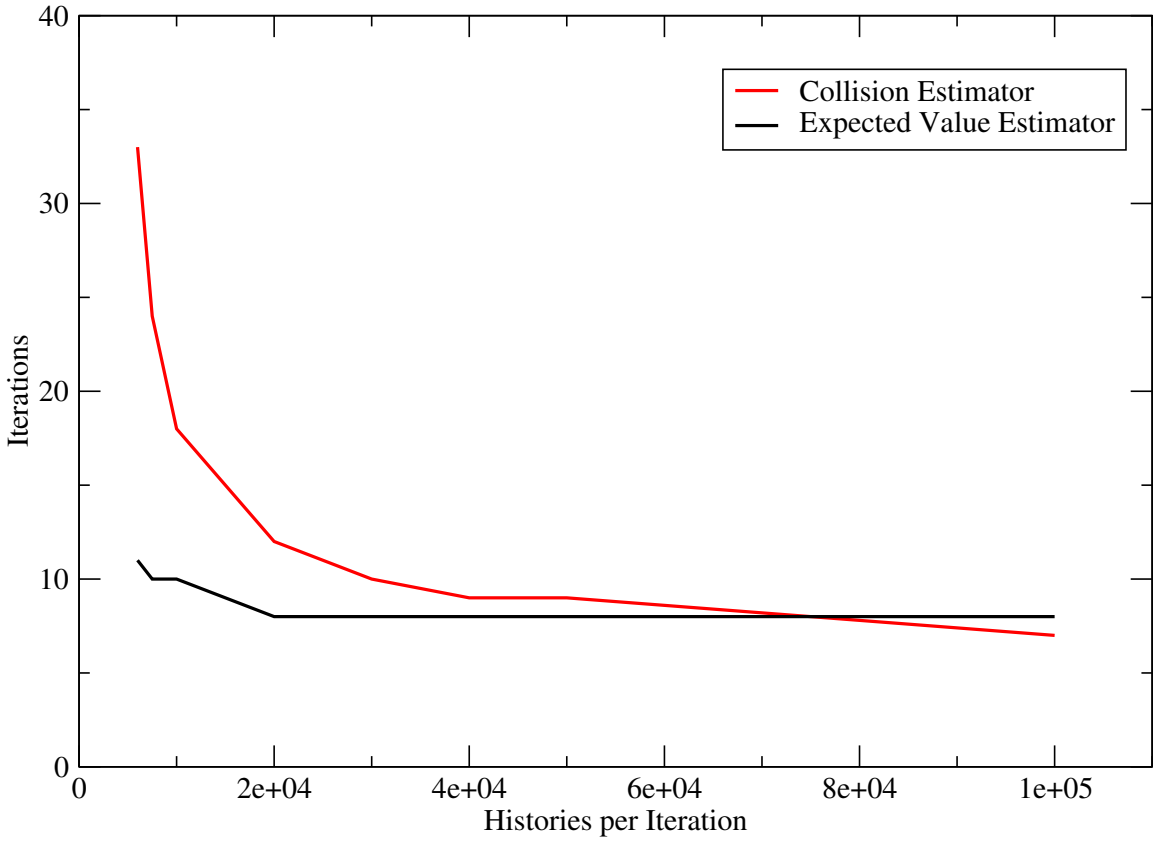

Figure 2.11: **Iterations (s) to converge vs. Monte Carlo histories per MCSA iteration for a 200×200 square mesh and a weight cutoff of 1×10^{-4}.** *For low numbers of histories, the expected value estimator performance is significantly better than the collision estimator. At higher numbers of histories, the estimators become roughly equivalent.*

more important to MCSA convergence than those that occur later and with lower weight contributions. To further motivate using a larger weight cutoff, Figure 2.14 gives the CPU time need to converge as a function of weight cutoff. As expected, lowering the weight cutoff lengthens the random walk lengths and increases CPU time at no gain of iterative performance. In general, these results suggest using the expected value estimator over the collision estimator with MCSA as well as using a larger weight cutoff.

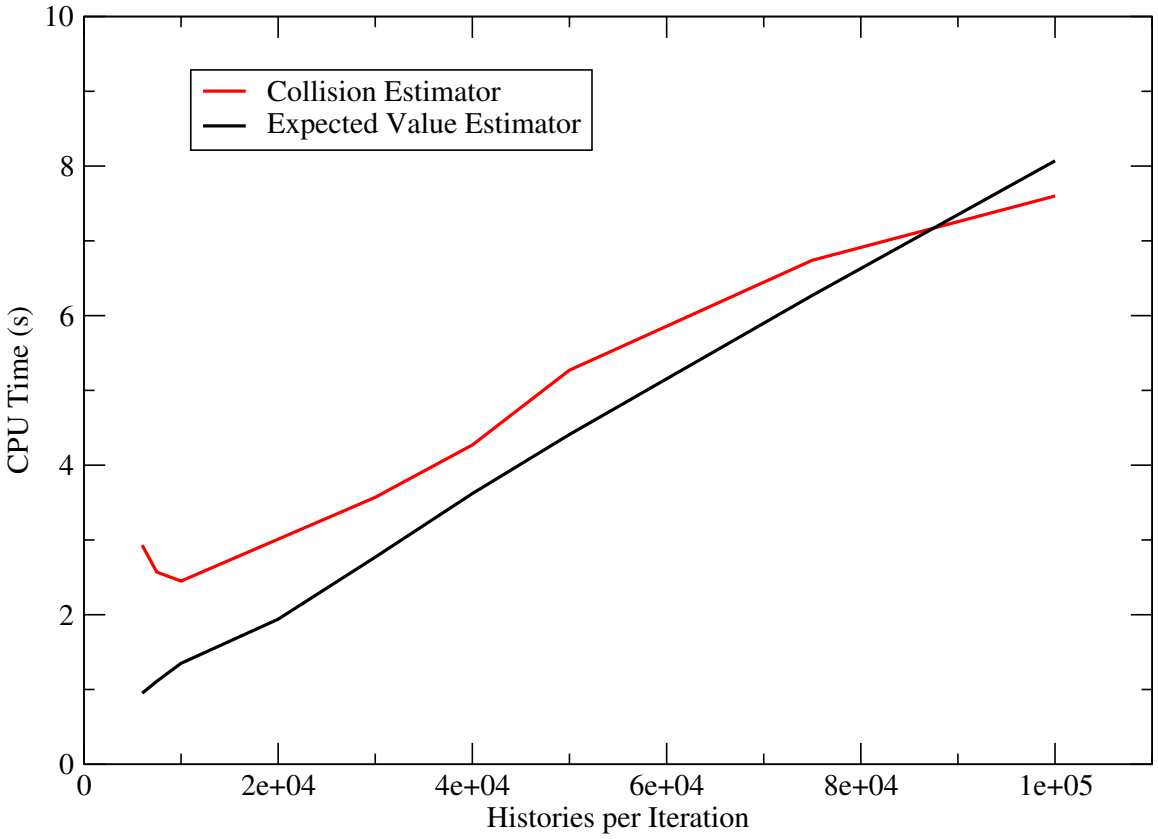

Figure 2.12: **CPU Time (s) to converge vs. Monte Carlo histories per MCSA iteration for a 200×200 square mesh and a weight cutoff of 1×10^{-4}.** *For low numbers of histories, the expected value estimator performance is better than the collision estimator due to a lower iteration count while the actual compute time per iteration is higher.*

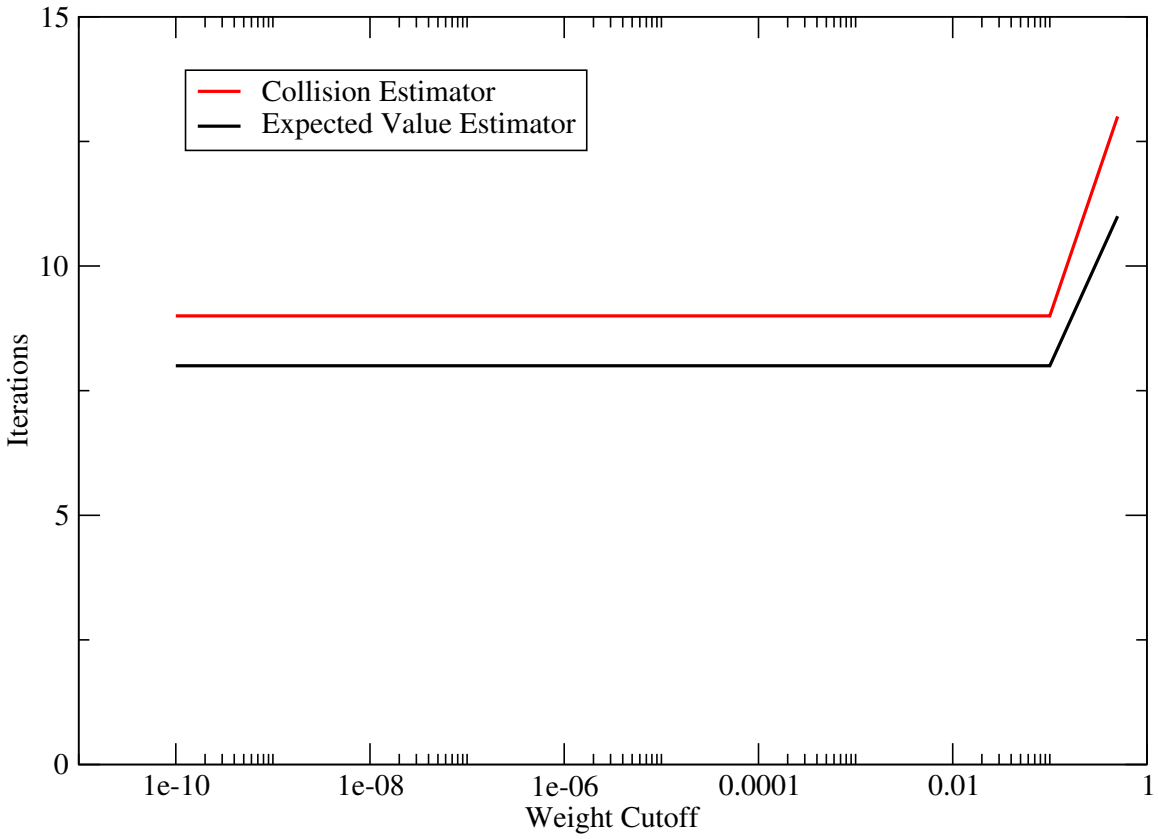

Figure 2.13: **Iterations (s) to converge vs. history weight cutoff for a** 200×200 **square mesh and 40,000 histories.** *For low numbers of histories, the expected value estimator performance is significantly better than the collision estimator. At higher numbers of histories, the estimators become roughly equivalent.*

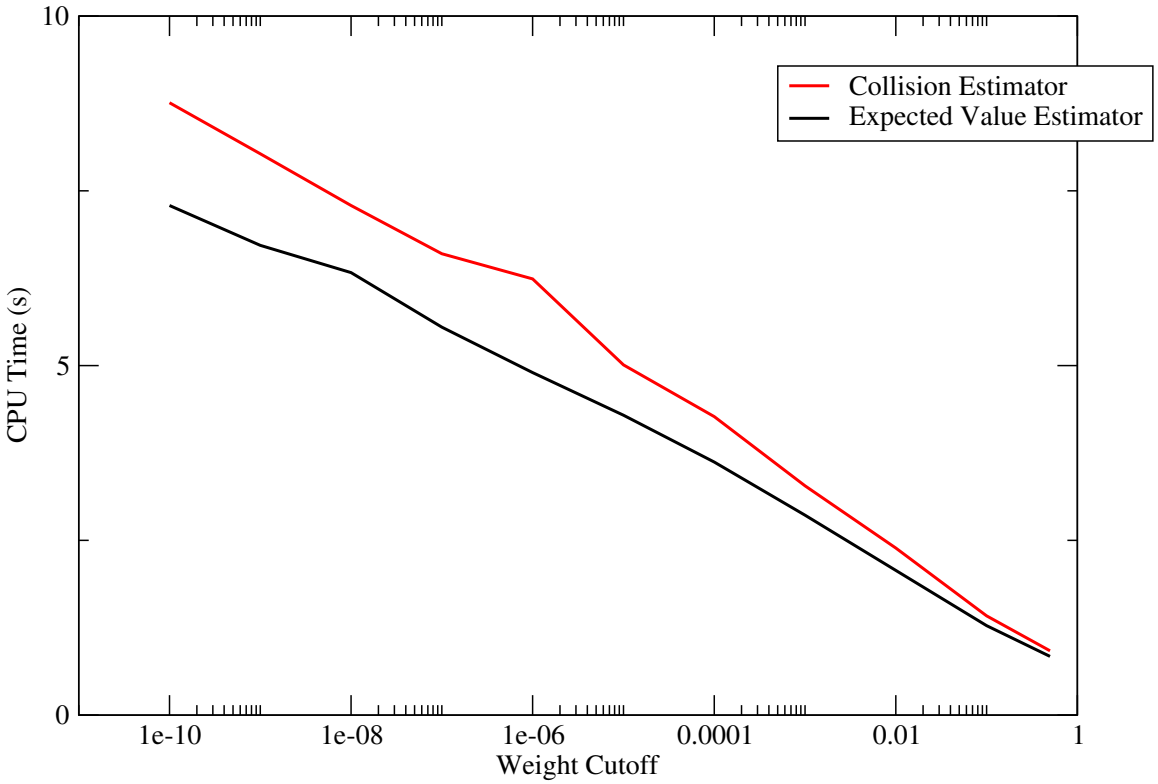

Figure 2.14: **CPU Time (s) to converge vs. history weight cutoff for a 200×200 square mesh and 40,000 histories.** *For low numbers of histories, the expected value estimator performance is better than the collision estimator due to a lower iteration count while the actual compute time per iteration is higher.*

2.10 Variance Reduction Techniques

Variance reduction is a mechanism by which the probabilities and weights of the Monte Carlo problem are modified to improve the performance of a given estimator. In many cases, such modifications will introduce bias into the system. However, using Monte Carlo within an MCSA iterative scheme provides a potential buffer for the solution from this bias as the correction generated by the Monte Carlo solve will contain large statistical error even without variance reduction. For this work, any method that alters the weights and probabilities of the Monte Carlo method with the aim of improving either iterative performance or time to convergence will be defined as a variance reduction scheme.

2.10.1 Artificial Absorption

The random walk sequences generated by the adjoint Neumann Ulam decomposition in Eqs (2.44) and (2.45) will continue on indefinitely without the implementation of a weight cutoff procedure (which in itself is effectively a form of biased variance reduction). This is due to the fact that all states to which a stochastic history may move in a given transition exist within the system and have a non-zero weight. A preliminary variance reduction scheme is to introduce *artificial absorption* into the system such that at each transition event, a stochastic history will now have some finite probability of being terminated through absorption as well as transition to other states in the system. This probability, p_{abs}, modifies the probabilities and weights in the adjoint random walk sequence as:

$$p_{ij} = \frac{|h_{ji}|}{\sum_j |h_{ji}|}(1 - p_{abs}) \,, \tag{2.81}$$

and

$$w_{ij} = \frac{sign(h_{ji})}{(1 - p_{abs})} \sum_j |h_{ji}| \,. \tag{2.82}$$

At each transition step, a history at state i will then sample the probability distribution function generated by Eq (2.81) with an additional absorption state added to the distribution. If the sampling procedure results in a transition to the absorption state, the history is immediately terminated. The advantages of this for residual Monte Carlo are that increasing the absorption probability decreases the length of each random walk (resulting in speedup), while the random walk has a higher weight contribution at every step. This higher weight means that each history is depositing more information

near its birth site which will on average be located where the system residual is the largest. An additional advantage of this formulation is that the full Neumann-Ulam decomposition is maintained.

To observe the effects of this variance reduction on MCSA solutions, the transient Poisson problem was again solved with the adjoint method, this time with varying values of artificial absorption. A 200×200 grid was used with 40,000 histories at every iteration with the expected value estimator converged to a tolerance of 1×10^{-8}. To reduce the effect of the weight cutoff on this study, the weight cutoff was set to 1×10^{-12} such that it is significantly more likely that a history will be terminated by absorption rather than weight cutoff. Figure 2.15 gives the number of iterations to converge as a function of the artificial absorption probability. Up to a probability of roughly 0.5, the iterative performance is not significantly affected with the information lost due to shortened random walks causing a small increase in the number of iterations. At a probability of 0.6, the number of iterations required grows rapidly as the bias creates an MCSA correction that pushes the solution in the wrong direction. Convergence was observed to be lost at probabilities of 0.7 and higher.

The random walk length has a strong effect on CPU time as well. Figure 2.16 gives the CPU time in seconds required to converge as a function of artificial absorption probability. Initially, small absorption probabilities not only maintain iterative performance, but also drastically reduced the time required to converge as absorption was more frequent than even a large weight cutoff but the weights themselves were not significantly modified. As the number of iterations grow rapidly, so does the time required to converge to a solution. Based on these results, using a small amount of artificial absorption should provide some CPU speedup while maintaining iterative performance. For the problem presented here, an absorption probability of around 0.25 may be the best choice as it minimizes CPU time.

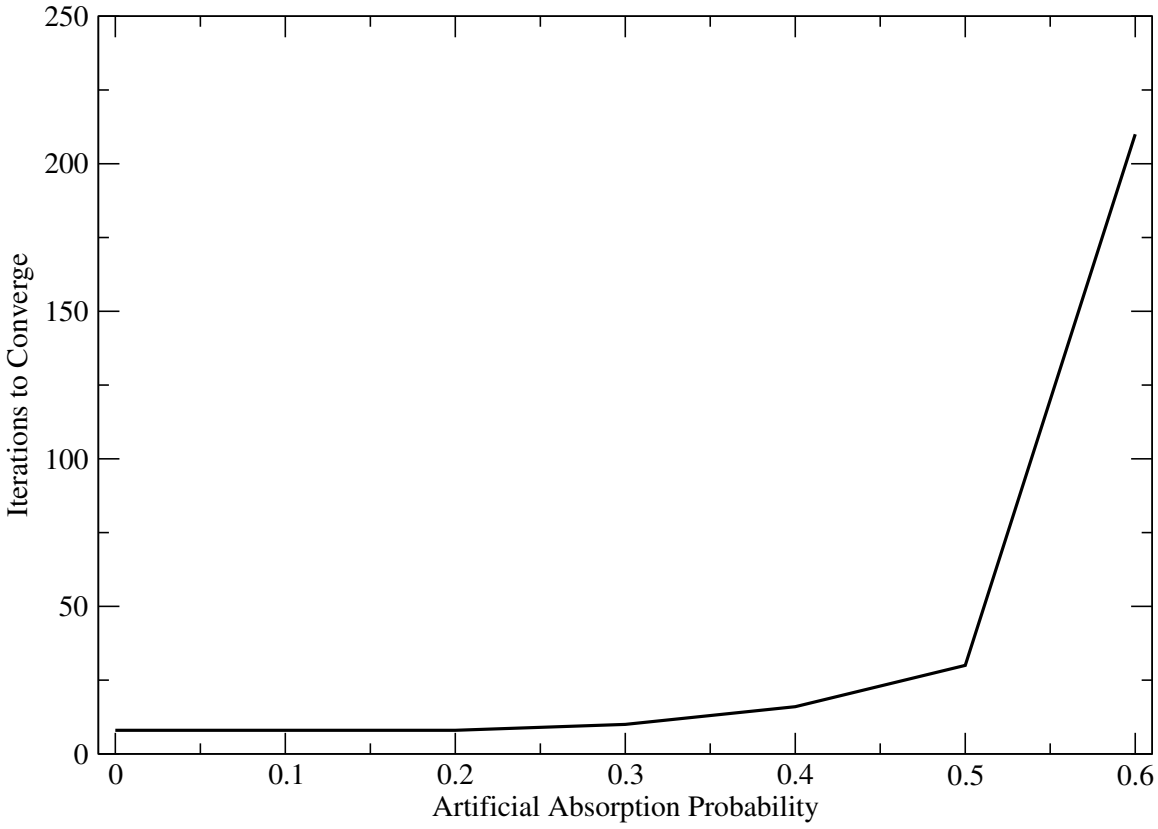

Figure 2.15: **Iterations (s) to converge vs. artificial absorption probability for a 200×200 square mesh and 40,000 histories.** *The addition of absorption is detrimental to iterative performance due to both the shortening of the random walks as well as the modification of the transition weight.*

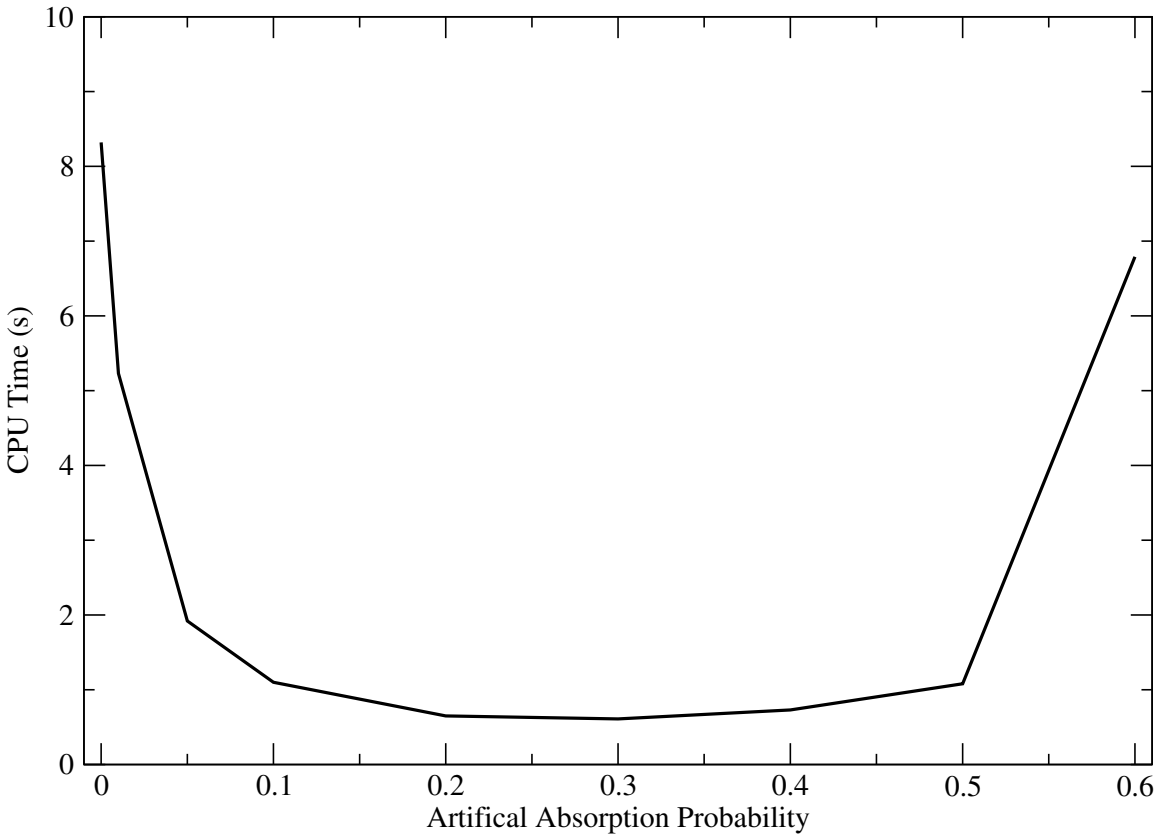

Figure 2.16: **CPU Time (s) to converge vs. artificial absorption probability for a** 200×200 **square mesh and 40,000 histories.** *The reduced random walk length speeds up the calculation with a fixed number of histories. Too much artificial absorption increases iterations rapidly giving an increasing CPU time.*

2.10.2 Reduced Domain Approximation

For scalable parallel implementations and efficient Monte Carlo transport, the domain is required to be sparse. We may find through general preconditioning operations as outlined by the Monte Carlo problem in Eq (2.75) that this sparsity is lost. To alleviate this, we propose modifying the domain sampled by the Monte Carlo method to generate the correction within MCSA by removing terms in the iteration matrix through some predetermined criteria in order to recover sparsity. As the Neumann-Ulam scheme is a stochastic realization of multiple matrix-vector multiplies, the criteria we choose should be based on maintaining those elements that will make the largest contributions to the multiplication and therefore recover as much of the original product as possible.

For each row in the preconditioned system, we can then apply the reduced domain approximation to eliminate those elements that either fall below a specified tolerance level, keep the N largest elements in each row, or both giving:

$$\mathbf{x}^{k+1/2} = \mathbf{x}^{k} + \mathbf{r}^{k} \,, \tag{2.83a}$$

$$\mathbf{r}^{k+1/2} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1/2} \,, \tag{2.83b}$$

$$\hat{\mathbf{A}}\delta\mathbf{x}^{k+1/2} = \mathbf{r}^{k+1/2} \,, \tag{2.83c}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta\mathbf{x}^{k+1/2} \,, \tag{2.83d}$$

$$\mathbf{r}^{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1} \,, \tag{2.83e}$$

where $\hat{\mathbf{A}}$ in Eq (2.83c) means that the reduced domain approximation has been applied to the iteration matrix in the Monte Carlo method. If enough of the terms in the original iteration matrix have been maintained, the character of the original iteration matrix should be retained and continue to accelerate the iterative scheme. The effects of the reduced domain approximation on neutron transport systems where it is required due to the explicit preconditioning strategy will be presented in Chapter 3.

Chapter 3

Monte Carlo Synthetic Acceleration Methods for the SP_N Equations

In this chapter, we briefly derive the simplified P_N (SP_N) equations, closely following the work of Evans (Evans, 2013)[1], in order to gain full understanding of the underlying system and its potential behavior in a Monte Carlo context. From the P_N equations, we apply a set of approximations to yield the SP_N equations for fixed source and criticality problems. Using the fully-formed linear operator for the transport problem, we explore solutions to the SP_N equations with Monte Carlo Synthetic Acceleration using a light water reactor fuel assembly criticality calculation as the driving problem. Several difficulties arise when applying MCSA to these problems that were not observed when investigating the simpler transport systems.

For Jacobi-based preconditioners, convergence with MCSA is difficult if not impossible for ill-conditioned systems with this behavior demonstrated using a simple neutron diffusion problem. In order to effectively solve the SP_N equations for the fuel assembly problem, a suite of preconditioners along with a relaxation scheme is developed and studied within the context of MCSA. Using these preconditioners, several additional issues are observed and alleviated to a certain extent by applying the reduced domain approximation. MCSA solutions for the fuel assembly problem are then verified by comparing the solutions against production Krylov solvers for problems with varying energy groups. Finally performance is analyzed through comparison with those same Krylov solvers in terms of both iterative performance and CPU timing using the same set of problems utilized for the verification.

[1]The SP_N derivations in this chapter and the associated appendices are heavily based on those presented by Evans in (Evans, 2013)

3.1 The Neutron Transport Equation

As a starting point for the SP_N equations we define the time-independent neutron transport equation (Lewis, 1993):

$$\hat{\Omega} \cdot \vec{\nabla}\psi(\vec{r}, \hat{\Omega}, E) + \sigma(\vec{r}, E)\psi(\vec{r}, \hat{\Omega}, E) = \iint \sigma_s(\vec{r}, E' \to E, \hat{\Omega}' \cdot \hat{\Omega})\psi(\vec{r}, \hat{\Omega}', E')d\Omega' dE' + q(\vec{r}, \hat{\Omega}, E)\,, \quad (3.1)$$

with the variables defined as:

- $\vec{r}$ - neutron spatial position
- $\hat{\Omega}$ - neutron streaming direction with radial component μ and azimuthal component ω
- $\hat{\Omega}' \cdot \hat{\Omega} = \mu_0$ is the angle of scattering
- E - neutron energy
- $\psi(\vec{r}, \hat{\Omega}, E)$ - angular flux
- $\sigma(\vec{r}, E)$ - total interaction cross section
- $\sigma_s(\vec{r}, E' \to E, \hat{\Omega}')$ - probability of scattering from direction $\hat{\Omega}'$ into an angular domain $d\hat{\Omega}'$ about the direction $\hat{\Omega}$ and from energy E' to an energy domain dE' about energy E
- $q(\vec{r}, \hat{\Omega}, E)$ - external source of neutrons.

For this work, it is sufficient to formulate Eq (3.1) in 1-dimensional Cartesian geometry:

$$\mu\frac{\partial}{\partial x}\psi(x, \mu, E) + \sigma(x, E)\psi(x, \mu, E) = \iint \sigma_s(x, E' \to E, \hat{\Omega}' \cdot \hat{\Omega})\psi(x, \hat{\Omega}', E')d\Omega' dE' + \frac{q(x, E)}{4\pi}\,, \quad (3.2)$$

where the angular component of the solution is no longer dependent on the azimuthal direction of travel and an isotropic source of neutrons is assumed. In addition, for

fission systems the eigenvalue form of the transport equation is:

$$\hat{\Omega}\cdot\vec{\nabla}\psi(\vec{r},\hat{\Omega},E)+\sigma(\vec{r},E)\psi(\vec{r},\hat{\Omega},E)= \iint\sigma_s(\vec{r},E'\rightarrow E,\hat{\Omega}'\cdot\hat{\Omega})\psi(\vec{r},\hat{\Omega}',E')d\Omega'dE'+ \frac{1}{k}\chi(E)\iint\nu\sigma_f(\hat{r},E')\psi(\vec{r},\hat{\Omega}',E')d\Omega'dE'+q(\vec{r},\hat{\Omega},E)\,, \quad (3.3)$$

with the additional variables defined as

- k - multiplication factor
- $\chi(E)$ - fission neutron energy spectrum
- ν - average number of neutrons per fission
- $\sigma_f(r,E')$ - fission cross section .

In 1-dimensional Cartesian geometry, Eq (3.3) becomes:

$$\mu\frac{\partial}{\partial x}\psi(x,\mu,E)+\sigma(x,E)\psi(x,\mu,E)= \iint\sigma_s(x,E'\rightarrow E,\hat{\Omega}'\cdot\hat{\Omega})\psi(x,\hat{\Omega}',E')d\Omega'dE'+ \frac{1}{k}\chi(E)\iint\nu\sigma_f(x,E')\psi(x,\hat{\Omega}',E')d\Omega'dE'+\frac{q(x,E)}{4\pi}\,. \quad (3.4)$$

3.2 Derivation of the Monoenergetic SP_N Equations

The P_N equations as derived in Appendix B give $N+1$ coupled first-order equations capturing the spatial and angular-dependence of the solution. In multiple dimensions, the equation set becomes large and coupled not only through angular moments but also through the spatial variables. As a simpler alternative to multidimensional P_N solutions, Gelbard recognized in 1960 that the planar P_N equations could be simplified and applied an ad-hoc method to extend them to multiple dimensions, yielding the SP_N equations. These equations are not only fewer in number, but also take on a diffusion-like form while maintaining the angular character of the flux, making them amenable to solutions with modern diffusion methods.

First, the P_N equations can be simplified to $(N+1)/2$ second-order equations by solving for the n^{th} Legendre flux moment in the odd-order equations:

$$\phi_n = \frac{1}{\Sigma_n}\left[q\delta_{no} - \frac{\partial}{\partial x}\left(\frac{n}{2n+1}\phi_{n-1} + \frac{n+1}{2n+1}\phi_{n+1}\right)\right], \tag{3.5}$$

for $n = 1, 3, \cdots, N$ and $\delta_{no} = 0\ \forall n \neq 0$. We can insert the odd moments into Eq (B.31) to get a reduced group of equations for the even moments:

$$\begin{aligned} -\frac{\partial}{\partial x}\Bigg[\frac{n}{2n+1}\frac{1}{\Sigma_{n-1}}\frac{\partial}{\partial x}\left(\frac{n-1}{2n-1}\phi_{n-2} + \frac{n}{2n-1}\phi_n\right) \\ + \frac{n+1}{2n+1}\frac{1}{\Sigma_{n+1}}\frac{\partial}{\partial x}\left(\frac{n+1}{2n+3}\phi_n + \frac{n+2}{2n+3}\phi_{n+2}\right)\Bigg] \\ + \Sigma_n\phi_n = q\delta_{n0} \qquad n = 0, 2, 4, \cdots, N\,. \end{aligned} \tag{3.6}$$

Immediately, we note the diffusion-like nature of Eq (3.6) as compared to the original P_N equations. To extend these equations to multiple dimensions, Gelbard simply replaced the planar spatial derivatives in the reduced set of equations with general multidimensional gradient operators:

$$\begin{aligned} -\nabla\cdot\Bigg[\frac{n}{2n+1}\frac{1}{\Sigma_{n-1}}\nabla\left(\frac{n-1}{2n-1}\phi_{n-2} + \frac{n}{2n-1}\phi_n\right) \\ + \frac{n+1}{2n+1}\frac{1}{\Sigma_{n+1}}\nabla\left(\frac{n+1}{2n+3}\phi_n + \frac{n+2}{2n+3}\phi_{n+2}\right)\Bigg] \\ + \Sigma_n\phi_n = q\delta_{n0} \qquad n = 0, 2, 4, \cdots, N\,, \end{aligned} \tag{3.7}$$

yielding a multidimensional set of $(N+1)/1$ angular coupled equations defined as the SP_N equations. As with the P_N equations, we provide closure to this set of equations

with $\phi_{N+1} = 0$. As a concrete example, we will consider the SP_7 equations:

$$-\nabla \cdot \frac{1}{3\Sigma_1}\nabla(\phi_0 + 2\phi_2) + \Sigma_0\phi_0 = q \tag{3.8a}$$

$$-\nabla \cdot \left[\frac{2}{15\Sigma_1}\nabla(\phi_0 + 2\phi_2) + \frac{3}{35\Sigma_3}\nabla(3\phi_2 + 4\phi_4)\right] + \Sigma_2\phi_2 = 0 \tag{3.8b}$$

$$-\nabla \cdot \left[\frac{4}{63\Sigma_3}\nabla(3\phi_2 + 4\phi_4) + \frac{5}{99\Sigma_5}\nabla(5\phi_4 + 6\phi_6)\right] + \Sigma_4\phi_4 = 0 \tag{3.8c}$$

$$-\nabla \cdot \left[\frac{6}{143\Sigma_5}\nabla(5\phi_4 + 6\phi_6) + \frac{7}{195\Sigma_7}\nabla(7\phi_6)\right] + \Sigma_6\phi_6 = 0\,. \tag{3.8d}$$

To further modify these equations, we can use a change of variables to create a new group of equations such that the gradients are operating on a single vector:

$$u_1 = \phi_0 + 2\phi_2 \tag{3.9a}$$

$$u_2 = 3\phi_2 + 4\phi_4 \tag{3.9b}$$

$$u_3 = 5\phi_4 + 6\phi_6 \tag{3.9c}$$

$$u_4 = 7\phi_6\,. \tag{3.9d}$$

When substituted into Eq (3.8), these terms give:

$$-\nabla \cdot \frac{1}{3\Sigma_1}\nabla u_1 + \Sigma_0\left[u_1 - \frac{2}{3}u_2 + \frac{8}{15}u_3 - \frac{16}{35}u_4\right] = -q \tag{3.10a}$$

$$-\nabla \cdot \left[\frac{2}{15\Sigma_1}\nabla u_1 + \frac{3}{35\Sigma_3}\nabla u_2\right] + \Sigma_2\left[\frac{1}{3}u_2 - \frac{4}{15}u_3 + \frac{8}{35}u_4\right] = 0 \tag{3.10b}$$

$$-\nabla \cdot \left[\frac{4}{63\Sigma_3}\nabla u_2 + \frac{5}{99\Sigma_5}\nabla u_3\right] + \Sigma_4\left[\frac{1}{5}u_3 - \frac{6}{35}u_4\right] = 0 \tag{3.10c}$$

$$-\nabla \cdot \left[\frac{6}{143\Sigma_5}\nabla u_3 + \frac{7}{195\Sigma_7}\nabla u_4\right] + \Sigma_6\left[\frac{1}{7}u_4\right] = 0\,. \tag{3.10d}$$

If we rearrange Eq (3.10) such that only one divergence operation is present in each equation, we can formulate this as a matrix system of 4 equations in the case of the SP_7 approximation:

$$-\nabla \cdot D_n\nabla u_n + \sum_{m=1}^{4} A_{nm}u_m = q_n \qquad n = 1, 2, 3, 4\,, \tag{3.11}$$

with $\mathbf{u}$ the vector of solution variables:

$$\mathbf{u} = (u_1 \;\; u_2 \;\; u_3 \;\; u_4)^T \,, \tag{3.12}$$

$\mathbf{D}$ the vector of effective diffusion coefficients:

$$\mathbf{D} = \left(\frac{1}{3\Sigma_1} \;\; \frac{1}{7\Sigma_3} \;\; \frac{1}{11\Sigma_5} \;\; \frac{1}{15\Sigma_7} \right)^T \,, \tag{3.13}$$

$\mathbf{q}$ the vector of source terms where the 0^{th} moment source has now been distributed through the system:

$$\mathbf{q} = (q \;\; -\frac{2}{3}q \;\; \frac{8}{15}q \;\; -\frac{16}{35}q)^T \,, \tag{3.14}$$

and $\mathbf{A}$ a matrix of angular scattering terms:

$$\mathbf{A} = \begin{bmatrix} (\Sigma_0) & (-\frac{2}{3}\Sigma_0) & (\frac{8}{15}\Sigma_0) & (-\frac{16}{35}\Sigma_0) \\ (-\frac{2}{3}\Sigma_0) & (\frac{4}{9}\Sigma_0 + \frac{5}{9}\Sigma_2) & (-\frac{16}{45}\Sigma_0 - \frac{4}{9}\Sigma_2) & (\frac{32}{105}\Sigma_0 + \frac{8}{21}\Sigma_2) \\ (\frac{8}{15}\Sigma_0) & (-\frac{16}{45}\Sigma_0 - \frac{4}{9}\Sigma_2) & (\frac{64}{225}\Sigma_0 + \frac{16}{45}\Sigma_2 + \frac{9}{25}\Sigma_4) & (-\frac{128}{525}\Sigma_0 - \frac{32}{105}\Sigma_2 - \frac{54}{175}\Sigma_4) \\ (-\frac{16}{35}\Sigma_0) & (\frac{32}{105}\Sigma_0 + \frac{8}{21}\Sigma_2) & (-\frac{128}{525}\Sigma_0 - \frac{32}{105}\Sigma_2 - \frac{54}{175}\Sigma_4) & (\frac{256}{1225}\Sigma_0 + \frac{64}{245}\Sigma_2 + \frac{324}{1225}\Sigma_4 + \frac{13}{49}\Sigma_6) \end{bmatrix} . \tag{3.15}$$

Note that the term $\sum_{m=1}^{4} A_{nm} u_m$ in Eq (3.11) couples the moments in each equation while the diffusive term in each equation is only for a single 'pseudo-moment' u_n. As noted by Evans (Evans, 2013), lower order SP_N approximations can be generated by setting higher order even moments in this system to zero (e.g. $\phi_6 = \phi_4 = 0$ yields the SP_3 equations). Boundary conditions for these equations are provided in Appendix D, the multigroup form is presented in Appendix C, and the spatial discretization in Appendix E.

3.2.1 Eigenvalue Form of the SP_N Equations

For criticality problems, the space, angle, and energy discretizations for the SP_N equations can be applied to the general eigenvalue form of the transport equation given by Eq (3.3). We can readily replace the fixed source in Eq (C.7) with the fission

source derived in § B.4:

$$
\begin{aligned}
-\nabla\cdot\Bigg[&\frac{n}{2n+1}\boldsymbol{\Sigma}_{\mathbf{n-1}}{}^{-1}\nabla\Big(\frac{n-1}{2n-1}\boldsymbol{\Phi}_{\mathbf{n-2}}+\frac{n}{2n-1}\boldsymbol{\Phi}_{\mathbf{n}}\Big)\\
&+\frac{n+1}{2n+1}\boldsymbol{\Sigma}_{\mathbf{n+1}}{}^{-1}\nabla\Big(\frac{n+1}{2n+3}\boldsymbol{\Phi}_{\mathbf{n}}+\frac{n+2}{2n+3}\boldsymbol{\Phi}_{\mathbf{n+2}}\Big)\Bigg]\\
&+\boldsymbol{\Sigma}_{\mathbf{n}}\boldsymbol{\Phi}_{\mathbf{n}}=\frac{1}{k}\mathbf{F}\boldsymbol{\Phi}_{\mathbf{n}}\delta_{n0}\qquad n=0,2,4,\cdots,N\,. \quad (3.16)
\end{aligned}
$$

with the fission matrix $\mathbf{F}$ given by Eq (B.46). We again apply the change of variables to yield the set of multigroup pseudo-moments $\mathbb{U}_n$ where now the application of the fission matrix to the moment vectors will be expanded into a set of block matrices in identical fashion to the scattering matrices:

$$
-\nabla\cdot\mathbb{D}_n\nabla\mathbb{U}_n+\sum_{m=1}^{4}\mathbb{A}_{nm}\mathbb{U}_m=\frac{1}{k}\sum_{m=1}^{4}\mathbb{F}_{nm}\mathbb{U}_n\qquad n=1,2,3,4\,. \quad (3.17)
$$

It should be noted here that with respect to a general MCSA scheme, the introduction of the fission in the system will only affect the source vector in the linear system. The linear operator, and therefore overall MCSA performance will be dictated by streaming and scattering as defined on the left-hand side.

3.3 Spectral Analysis of the SP_N Equations

Before we can move on to solving the SP_N equations with both conventional and Monte Carlo methods, we must first understand their spectral character to ensure that MCSA will be applicable. As presented in Chapter 2, the spectral radius of the iteration matrix must be less than unity to ensure convergence. Therefore, we will compute the spectral radius of the iteration matrix formed by the SP_N equations while parametrically varying the SP_N order, the P_N order, the number of energy groups, and upscattering/downscattering between energy groups. For each problem in the parameter study, the values given in Table 3.1 were fixed for each spatial location and energy group. Mesh element sizes were the same in all cardinal directions with reflecting conditions on each boundary. For the energy parameter, 1 and 10 energy groups were used with upscatter/downscatter varied for the 10 group case. For the downscatter cases, all groups could scatter to all lower energy groups with the cross sections given in Table 3.1 while for the upscatter case all groups could scatter to all

Parameter	Value
Mesh Element Size	1.0
Mesh Elements	$4 \times 4 \times 4$
Reflecting Boundaries	True
Materials	1
Source Strength	1.0
Total Cross Section	5.0
In-Group Cross Section	0.25
Downscatter Cross Section	1.0
Upscatter Cross Section	0.1
Eigenvalue Solver Tolerance	1.0×10^{-8}

Table 3.1: **Fixed parameters for the SP_N spectral radius parameter study.** *The parameters were fixed for each spatial location and energy group.*

other energy groups with different cross sections for upscatter and downscatter used. A uniform isotropic source of neutrons is given in each group as well.

The matrices generated by the discretization of this problem are sparse with a block-based form as dictated by Eq (C.8). Figure 3.1 gives the sparsity pattern for a monoenergetic SP_7 discretization while Figures 3.2 and 3.3 give the sparsity patterns for the same problem for 10 energy groups without and with upscatter respectively. For these figures, the parameters in Table 3.1 with the number of spatial elements reduced to $2 \times 2 \times 2$ to show the blocks generated by the discretization. We note a few key features of the sparsity plots. The first is that for multigroup problems without full upscatter and downscatter (i.e. Figure 3.2), the resulting matrix is asymmetric and therefore a linear solver that can handle asymmetric linear systems is required. Nearly all problems of interest will not have full upscattering or downscattering. Second we note the largely diagonal character of these systems, although the blocks from Eq (C.9) are readily apparent. Our first attempt at preconditioning this system will be to use the point Jacobi preconditioning from §2.6.2.1 due to this diagonal form.

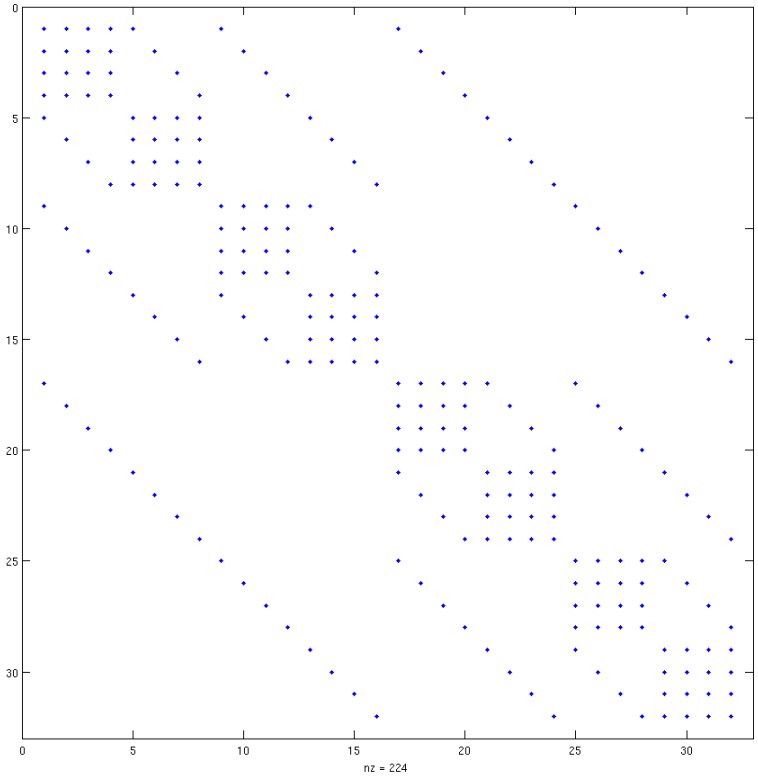

Figure 3.1: **Sparsity pattern for 1-group SP_7 discretization.** *A $2\times2\times2$ element mesh was used to show detail of the blocks formed by the discretization.*

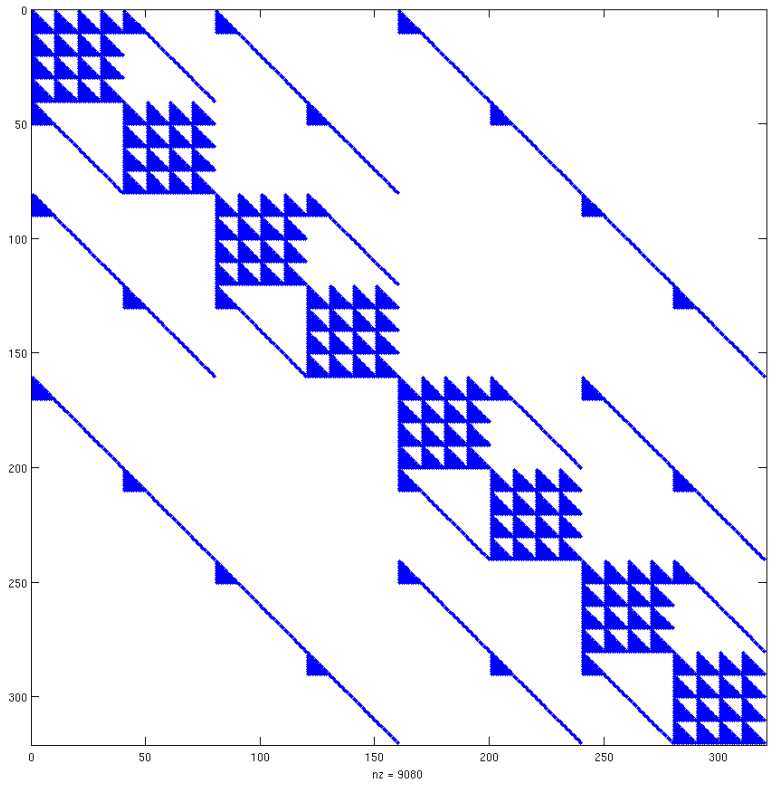

Figure 3.2: **Sparsity pattern for 10-group SP_7 discretization with downscatter only.** *A $2 \times 2 \times 2$ element mesh was used to show detail of the blocks formed by the discretization.*

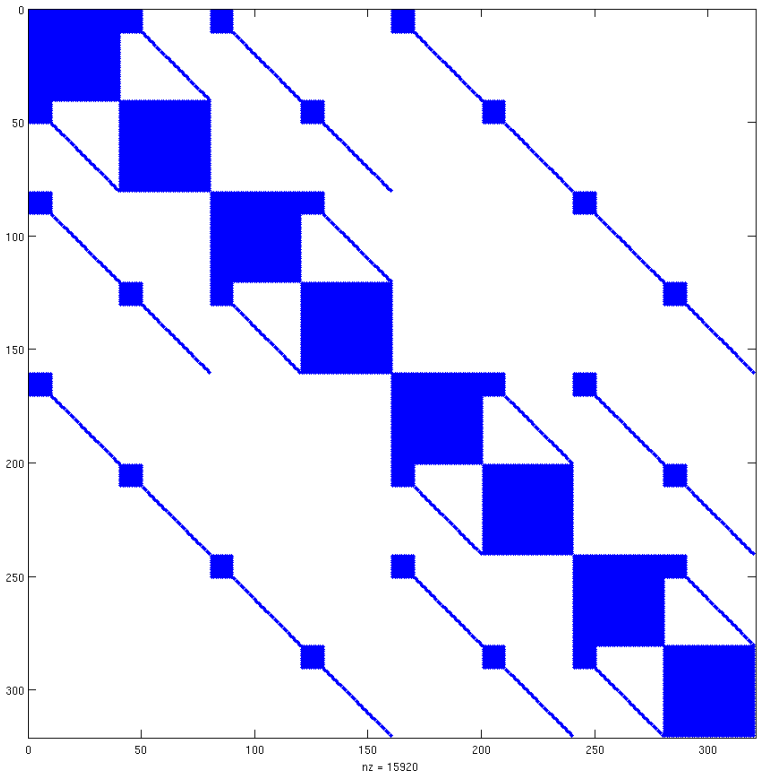

Figure 3.3: **Sparsity pattern for 10-group SP_7 discretization with downscatter and upscatter.** *A $2 \times 2 \times 2$ element mesh was used to show detail of the blocks formed by the discretization.*

3.3.1 Point Jacobi Spectral Analysis Results

Spectral radius computations were performed for the cases described above for the point Jacobi preconditioned iteration matrix $\mathbf{H} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$ with $\mathbf{M} = diag(\mathbf{A})$. Table 3.2 gives the results for the 1-group case, Table 3.3 for the 10-group case with full downscatter only and Table 3.4 for the 10-group case with full downscatter and full upscatter. It is readily apparent from the tabulated data that point Jacobi

		SP_N Order			
		1	**3**	**5**	**7**
	0	0.0635	0.6722	1.3144	1.976
	1	0.0666	0.6728	1.3141	1.9755
P_N Order	**3**	0.0666	0.6822	1.3141	1.9755
	5	0.0666	0.6822	1.3278	1.9847
	7	0.0666	0.6822	1.3278	1.9917

Table 3.2: **Spectral radius results for the point Jacobi preconditioned iteration matrix with 1 energy group.**

		SP_N Order			
		1	**3**	**5**	**7**
	0	0.0655	0.677	1.32	1.982
	1	0.071	0.6777	1.319	1.982
P_N Order	**3**	0.071	0.687	1.327	1.9872
	5	0.071	0.687	1.336	1.997
	7	0.071	0.687	1.336	1.9995

Table 3.3: **Spectral radius results for the point Jacobi preconditioned iteration matrix with 10 energy groups and full downscatter.**

preconditioning is insufficient. For problems of order SP_5 and SP_7, the method will not converge at all and for the upscatter case with SP_3 the spectral radius is still quite large for large P_N orders and therefore convergence is expected to be slow. These eigenvalues signal a need for a better preconditioning strategy to both ensure and improve convergence for Monte Carlo methods.

		SP_N Order			
		1	**3**	**5**	**7**
	0	0.7283	0.81	1.47	2.1446
	1	0.7317	0.8	1.46	2.1368
P_N Order	**3**	0.7317	0.91	1.526	2.2274
	5	0.7317	0.91	1.5344	2.2562
	7	0.7317	0.91	1.5345	2.2842

Table 3.4: **Spectral radius results for the point Jacobi preconditioned iteration matrix with 10 energy groups, full downscatter and full upscatter.**

3.3.2 Block Jacobi Preconditioning

If Monte Carlo methods are to be used to solve the SP_N system of equations, a different preconditioning strategy is required in order to ensure convergence for systems of all SP_N and P_N orders with arbitrary energy group structures. As another means of achieving this, we look back to the sparsity plots we generated in Figures 3.1, 3.2 and 3.3 as well as the multigroup SP_N equations. Initially, the diagonal character of the system led us to try point Jacobi preconditioning with only marginal results. From the sparsity plots we note the block structure that ultimately arises from the multigroup scattering matrices and their insertion into Eq (C.9). When full upscatter and downscatter are used the resulting blocks are completely dense while only downscatter gives a lower triangular scattering matrix and the block structure shown in Figure 3.2.

Based on this both block and diagonally dominant structure for matrices formed by the general multigroup SP_N equations, we instead choose *block Jacobi* preconditioning as a left preconditioner for the system. Like point Jacobi preconditioning, block Jacobi preconditioning extracts the diagonal elements of the matrix as the preconditioner where now the elements extracted are the blocks on the diagonal as shown on the left side of Figure 3.4. Shown on the right side of Figure 3.4, inversion of the preconditioner is trivial with each diagonal block inverted separately. For the SP_N equations, Eq (C.9) gives a block size of $N_g \times (N+1)/2$. The inversion of this preconditioner is trivial as shown on the right side of Figure 3.4. Each block can be inverted individually and combined to form the inverse. In addition, in the limit of a block size of one, the block Jacobi method reduces to the point Jacobi method. For high performance implementations this has several attractive properties. First, the blocks in the matrix come from the energy/angle discretization of the transport equation as given by Eq (C.9). Each block on the diagonal is bound to a mesh element in the system (note there are 8 blocks on the diagonal in each of the sparsity patterns with a mesh of $2 \times 2 \times 2$) and therefore we expect the matrix elements forming the block to be entirely local. Second, these blocks are typically dense and nearly lower triangular for many transport problems meaning that established dense matrix methods can be used for fast inversion.

3.3.3 Block Jacobi Spectral Analysis Results

Spectral radius computations for the block Jacobi preconditioned iteration matrix were performed for the same problems as the point Jacobi preconditioned case. Table 3.5

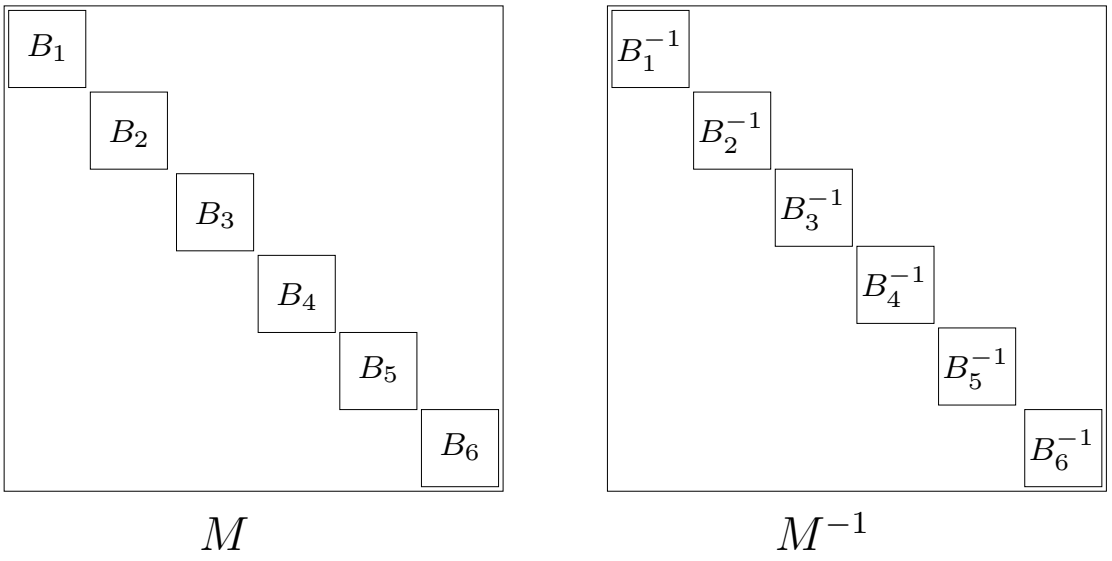

Figure 3.4: **Block Jacobi preconditioning strategy used for the SP_N equations.** *Left: The preconditioner is formed by the diagonal blocks of the matrix. Right: Inversion of the preconditioner is trivial and decoupled by block.*

gives the results for the 1-group case, Table 3.6 for the 10-group case with full downscatter only and Table 3.7 for the 10-group case with full downscatter and full upscatter. A block size of $N_g \times (N+1)/2$ was used for the preconditioner in all cases.

		SP_N Order			
		1	**3**	**5**	**7**
	0	0.0635	0.1269	0.1444	0.1513
	1	0.0666	0.1315	0.1474	0.1534
P_N Order	**3**	0.0666	0.1365	0.154	0.1592
	5	0.0666	0.1365	0.1562	0.163
	7	0.0666	0.1365	0.1562	0.164

Table 3.5: **Spectral radius results for the block Jacobi preconditioned iteration matrix with 1 energy group.**

From the tabulated block Jacobi data it is clear that this is a viable preconditioning choice for all SP_N problems in term of Monte Carlo solution methods. All cases were observed to have a spectral radius below unity and often significantly smaller than that, greatly improving convergence rates over the point Jacobi preconditioned problem. Based on these results, the block Jacobi method should be the first preconditioning

		SP_N Order			
		1	**3**	**5**	**7**
	0	0.0647	0.1275	0.1449	0.1514
	1	0.0686	0.1338	0.1484	0.1547
P_N Order	**3**	0.0687	0.1399	0.1582	0.1625
	5	0.0692	0.1399	0.1582	0.1657
	7	0.0678	0.1393	0.1624	0.166

Table 3.6: **Spectral radius results for the block Jacobi preconditioned iteration matrix with 10 energy groups and full downscatter.**

		SP_N Order			
		1	**3**	**5**	**7**
	0	0.1887	0.2267	0.2285	0.2286
	1	0.4535	0.5044	0.5045	0.5045
P_N Order	**3**	0.4535	0.6453	0.6506	0.6506
	5	0.4535	0.6453	0.6802	0.6818
	7	0.4535	0.6453	0.6802	0.6927

Table 3.7: **Spectral radius results for the block Jacobi preconditioned iteration matrix with 10 energy groups, full downscatter and full upscatter.**

strategy applied to more complicated problems representative of physical systems.

3.4 Fuel Assembly Criticality Calculations

Fuel assembly calculations are a critical piece of nuclear engineering infrastructure for reactor core analysis and design. At this level, individual fuel pins may be resolved at fine resolution in a variety of configurations. As a sophisticated problem of interest to push the limits of MCSA, a hot zero-power 17×17 pin assembly will be used with varying energy group structure and SP_N discretization in a criticality calculation. A cross section along the vertical axis showing homogenized fuel pin materials and the associated grid is given by Figure 3.5 while a cross section of the materials configuration along the horizontal axis is given by Figure 3.6. A detailed view of the assembly bottom is given in Figure 3.7. On the top and bottom of the assembly, vacuum conditions are used as well as on the top and right boundaries in Figure 3.5. Reflecting conditions are used on the left and bottom boundaries of Figure 3.5, effectively giving a representation of one quarter of the assembly. For the spatial discretization, each fuel pin (gray regions in Figure 3.5) is resolved by a 2×2 mesh with materials and cross sections homogenized over this region.

Significant geometric details are contained in the model including spacer grids, fuel pins with homogenized cladding and gas gap, core plenum, and moderator with boron. Group cross sections and other discrete nuclear data are generated as needed by a cross-section processing module dependent on the meshing parameters used to discretize the geometry and single-dimension pin-cell calculations for initial flux spectrum generation. Table 3.8 gives the primary design parameters for the fuel assembly calculations.

To generate the multiplication factor and steady-state flux distribution for this problem, at every eigenvalue iteration MCSA is used to solve the resulting SP_N problem using the provided fission source. Algorithm 3.1 presents the use of MCSA within a power iteration strategy to find the multiplication factor. Here, $\mathbf{M}$ is the

Algorithm 3.1 Power Iteration MCSA Scheme

$k_0 =$ initial guess
$\mathbf{\Phi}_0 =$ initial guess
$n = 0$
while $|\frac{k^n - k^{n-1}}{k^n}| < \epsilon$ **do** ▷ Iterate until convergence of the eigenvalue
  $\mathbf{M}\mathbf{\Phi}^{n+1} = \frac{1}{k^n}\mathbf{F}\mathbf{\Phi}^n$ ▷ Solve for the new flux state with MCSA
  $k^{n+1} = k^n \frac{\int \mathbf{F}\mathbf{\Phi}^{n+1} d\mathbf{r}}{\int \mathbf{F}\mathbf{\Phi}^n d\mathbf{r}}$ ▷ Update the multiplication factor
  $n = n + 1$
end while

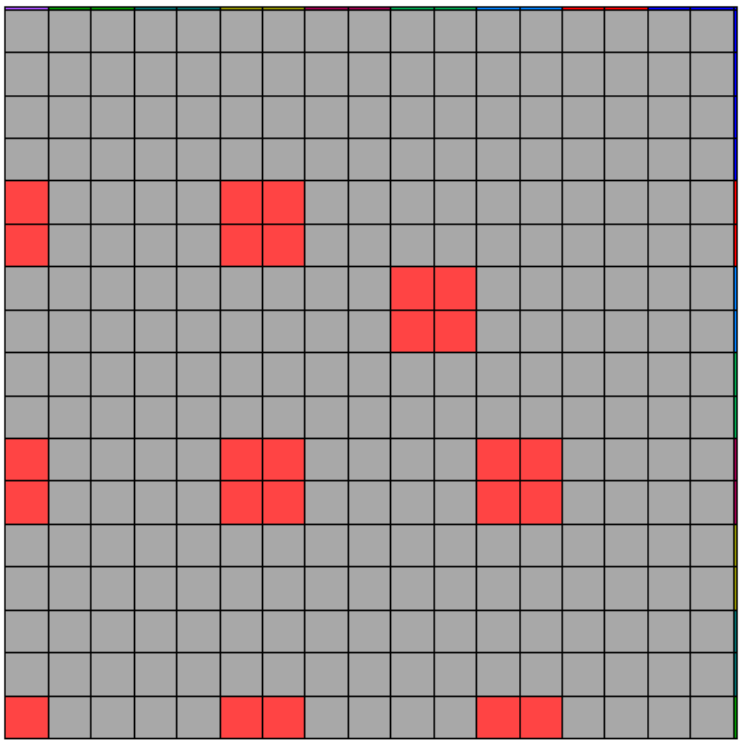

Figure 3.5: **Fuel assembly mesh and geometry cross section.** *Reflecting boundaries are used on the left and lower boundaries to create a complete* 17×17 *assembly geometry. Gray regions are homogenized fuel and red regions are homogenized moderator. Each fuel pin is resolved by a* 2×2 *mesh.*

Figure 3.6: **Fuel assembly geometry cross section.** *The geometry is subdivided along the axial direction into 50 zones spaced to capture material boundaries. Important details include spacer grids along the length of the fuel pins and reactor core structures on the top and bottom of the assembly. Lighter purple material in the center of the assembly is moderator and darker purple/red material is fuel.*

transport operator generated on the left-hand side of the SP_N discretization, $\mathbf{F}$ is the fission matrix, and $\mathbf{\Phi}$ the multigroup neutron flux. This problem is significantly more complicated than the simple test problem used for the previous spectral analysis. Fission has been introduced into the set of equations and the addition of moderator into the system will increase the amount of scattering, creating a significantly more difficult problem manifesting itself in an iteration matrix with a larger spectral radius.

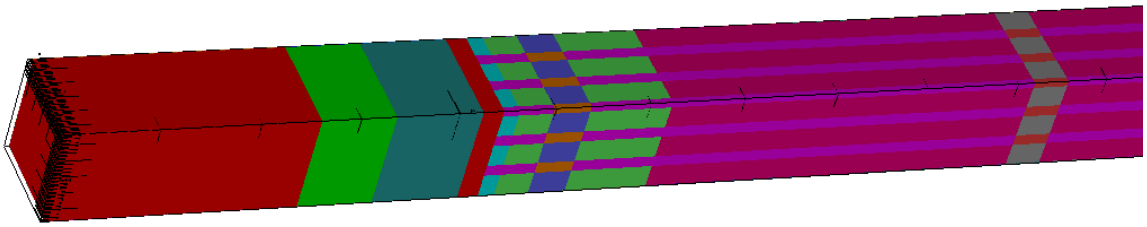

Figure 3.7: **Fuel assembly geometry end detail.** *Reactor core structure including spacer grids and plenum has been included. Lighter purple material on the right of the figure is moderator and darker purple/red material is fuel.*

Parameter	**Value**
Power Level	0 MW
Inlet Temperature	326.85C
Fuel Temperature	600C
Boron Concentration	1300 ppm
Moderator Density	0.743 g/cc
Helium Density	1.79×10^{-4} g/cc
Zirconium Density	6.56 g/cc
Stainless Steel Density	8.0 g/cc
Inconel Density	8.19 g/cc
UO2 Density	10.257 g/cc
Fuel Pin Radius (w/o clad)	0.4096 cm

Table 3.8: **Design parameters for the** 17×17 **pin fuel assembly criticality calculation.**

When using MCSA, the linear operator applied to $\mathbf{\Phi}^{n+1}$ at each eigenvalue iteration will dictate convergence and remain unchanged throughout the computation[2] while the addition of fission to the system will only modify the source of neutrons and the multiplication factor while not affecting Monte Carlo transport.

[2]The operator will change if, for example, physics feedback through temperature or potentially burnup is considered. These additional physics will modify the cross sections used to assemble $\mathbf{M}$. The calculations presented here will consist of a single eigenvalue calculation with a static $\mathbf{M}$.

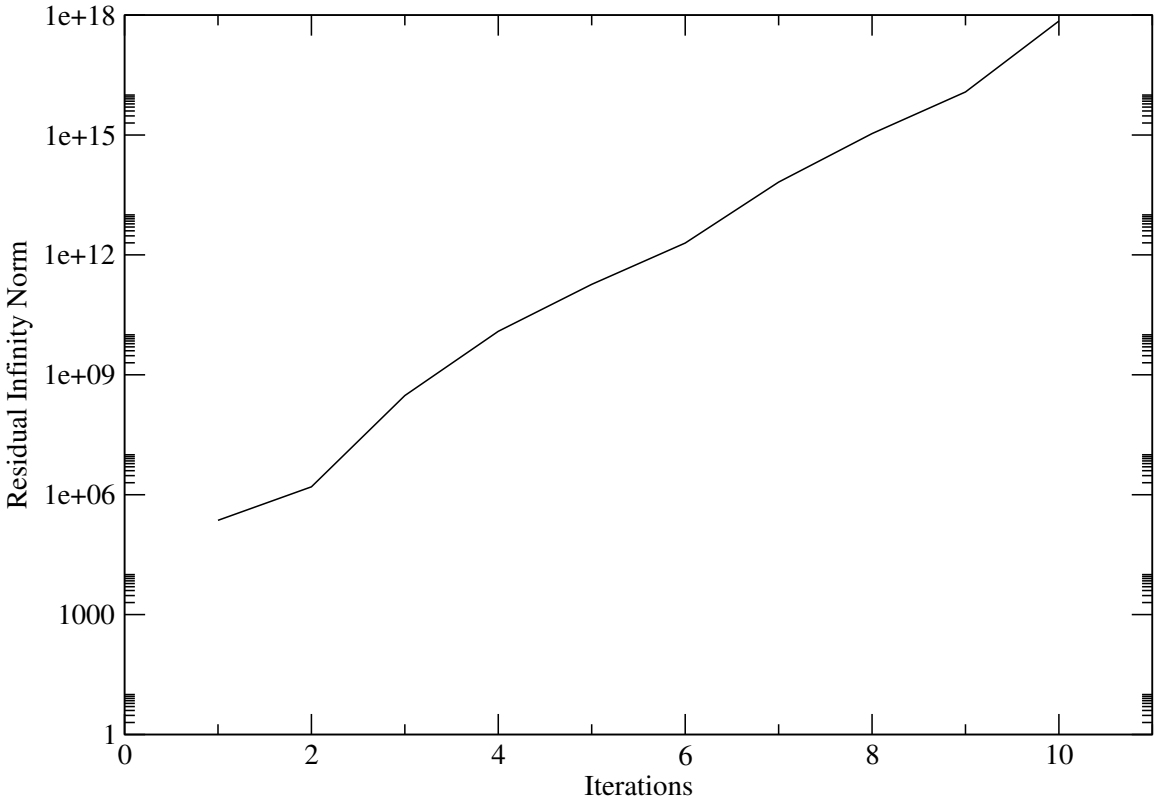

Figure 3.8: **Residual infinity norm vs. iteration for the block Jacobi preconditioned MCSA solve during the first eigenvalue iteration of the 1-group 17×17 fuel assembly problem.** *Convergence was not achieved with the block Jacobi preconditioned method.*

3.4.1 Preliminary Jacobi Preconditioned Calculations

Based on the success of block Jacobi preconditioning with the test problem used for the spectral radius parameter study, we use it first to solve the 17×17 fuel assembly problem. A single energy group problem was first solved with SP_1 discretization, effectively giving the one-speed neutron diffusion system for the fuel assembly resulting in 20,088 degrees of freedom in the problem. Figure 3.8 gives the residual infinity norm as a function of iteration for the MCSA linear solve in the first eigenvalue iteration using 25,000 stochastic histories at every iteration for the adjoint Neumann-Ulam solve. Convergence was not achieved as noted by the rapid rise in the residual over a few iterations. Based on the spectral radius computations performed, these results are not in line with expectations for this problem. Additional computations performed with 1×10^6 histories per iteration exhibited the same divergent behavior at a significant computational cost and compute time. Even if the problem may be ill-conditioned, we

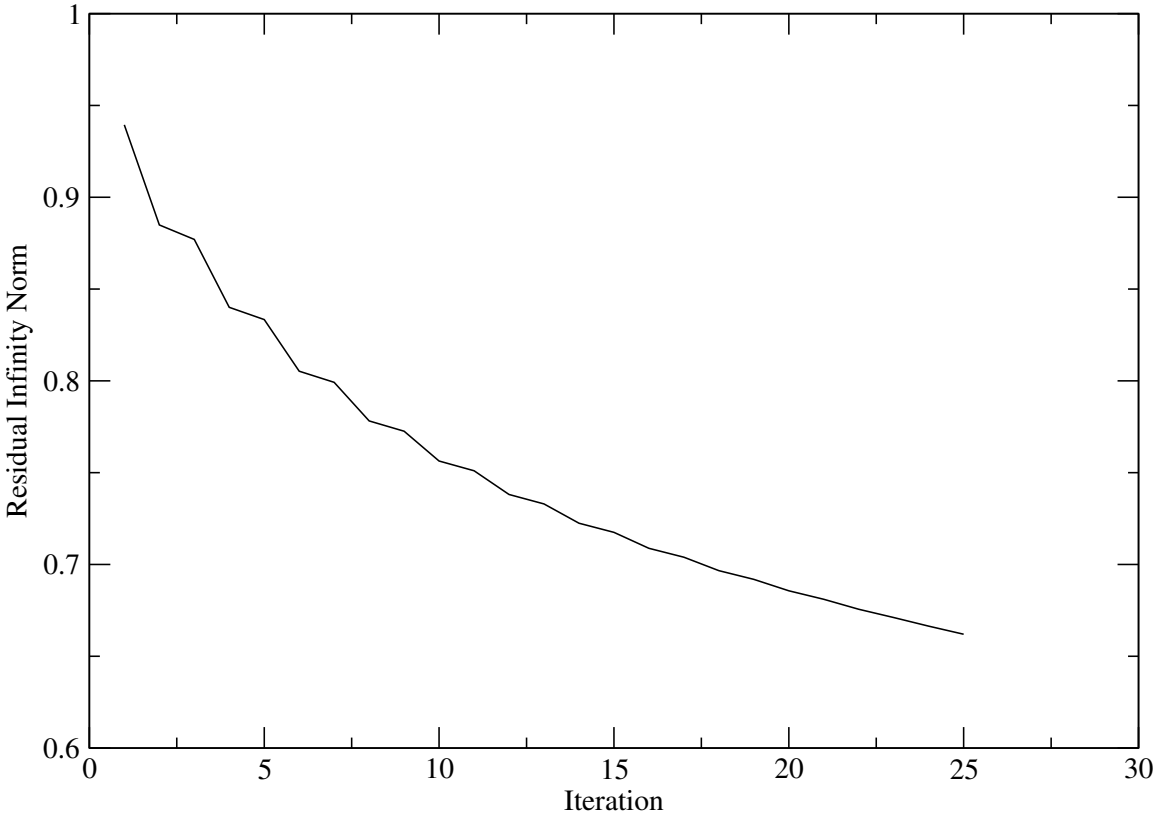

Figure 3.9: **Residual infinity norm vs. iteration for the block Jacobi preconditioned Richardson solve during the first eigenvalue iteration of the 1-group 17×17 fuel assembly problem.** *A spectral radius near 1 was observed for the iteration matrix.*

do expect convergence of MCSA. To investigate further, a block Jacobi preconditioned Richardson iteration was used to solve the same problem. Figure 3.9 gives the residual infinity norm as a function of iteration for the Richardson linear solve in the first eigenvalue iteration. Poor converge is observed for the Richardson iteration, however, convergence is achieved meaning that the preliminary eigenvalue criteria needed to satisfy MCSA convergence has also been met. The number of iterations required for the Richardson iteration to converge will give an approximation for the spectral radius via Eq (A.10). With 7291 iterations required to converge to a tolerance of 1×10^{-6}, $\rho(\mathbf{H}) \approx 0.998$, nearing the limits of MCSA applicability and well beyond the spectral radii generated in the initial spectral analysis.

Based on these results, it then appears that even if the simple criteria of a spectral radius of less than one is met and the Richardson iteration will converge with the same preconditioning, MCSA still may not converge. We then expect the issue to reside

with the Neumann-Ulam solve providing the correction as the Richardson iteration is known to provide the correct result. Furthermore, the fact that the spectral radius is less than one means that the stochastic histories in the block Jacobi preconditioned Neumann-Ulam method are eventually being terminated by the weight cutoff as no artificial absorption was used for these preliminary calculations. Based on this, the correction being generated by the Neumann-Ulam solve is the component of MCSA causing a divergent solution.

3.4.2 MCSA Breakdown

For the fuel assembly problem, the initial block Jacobi preconditioned[3] calculations used a one-speed SP_1 discretization, effectively giving a diffusion system. To study the breakdown of MCSA at iteration matrix spectral radii near one, we will use the simpler homogeneous 2-dimensional one-speed neutron diffusion system presented in Appendix F to isolate this behavior. In this system, we can vary the cross sections while maintaining a fixed grid in order to achieve varying spectral radii. For these studies, we neglect fission as MCSA behavior is dictated by the transport operator $\mathbf{M}$ in an eigenvalue scheme with the fission matrix used to generate a fixed source.

For each solver and estimator combination, the spectral radius of the iteration matrix generated by the diffusion problem was varied by changing the absorption cross section from 0.25 to 100 while fixing the grid size at 100×100 with $h = 0.1$ and a fixed scattering cross section of unity. For each parameter variation, a minimum of one stochastic history per degree of freedom (DOF) in the problem was used to compute the Monte Carlo correction. If the solver could not converge in less than 100 iterations, the number of histories was increased by increments of 5,000 until convergence was achieved in less than 100 iterations. The number of iterations required to converge MCSA and the time to converge was recorded as a means to capture the breakdown.

Figure 3.10 gives the number of iterations required to converge for the chosen number of histories per iteration given by Figure 3.11 using the adjoint solver with the collision and expected value estimators and the forward solver with the collision estimator. For spectral radii less than 0.97, all MCSA problems converged with 1 history per DOF (10,000 for this problem) with the number of iterations required to converge increasing as a function of spectral radius. Near a spectral radius of 0.97, the number of histories required to converge MCSA in less than 100 iterations takes a dramatic rise that exhibits neither exponential nor power law behavior. As the

[3]For 1-group SP_1 problems the block size is 1, giving a point Jacobi preconditioner.

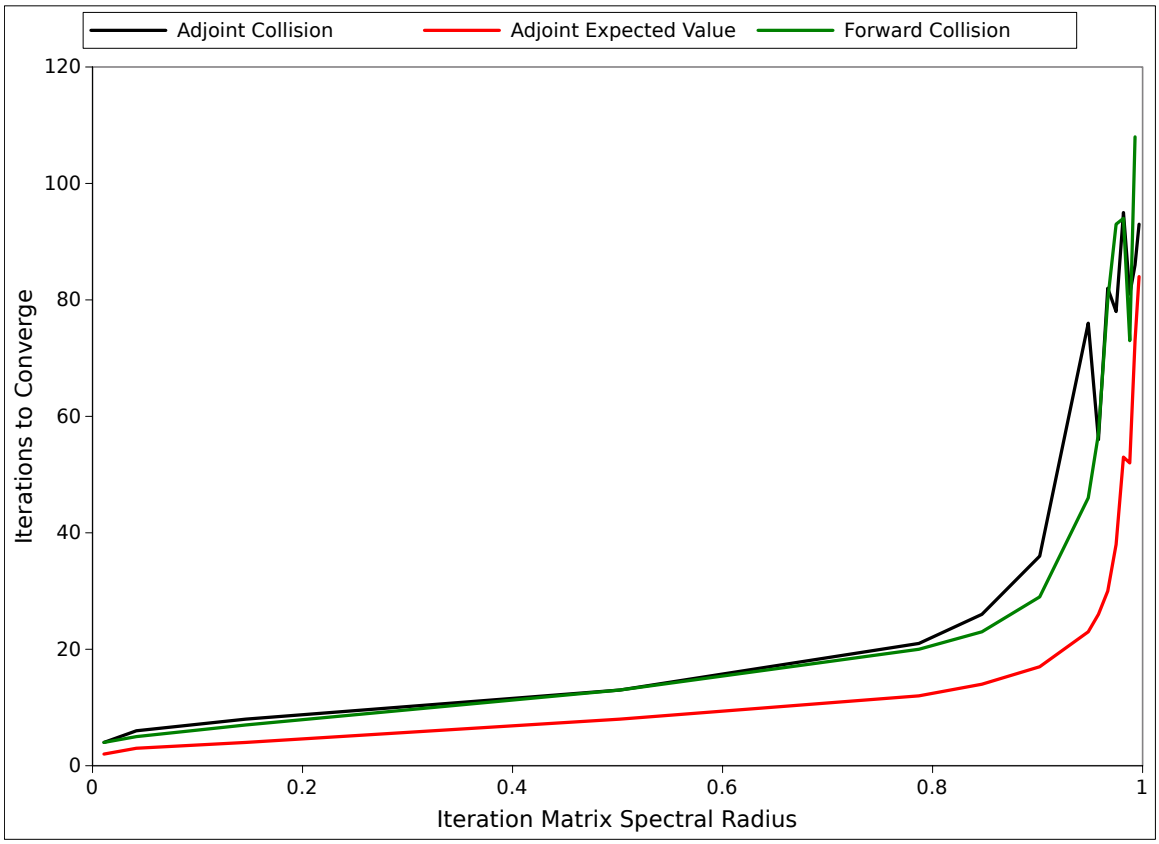

Figure 3.10: **Iterations required to converge as a function of spectral radius for the neutron diffusion problem.** *The number of histories was increased to achieve convergence in less than 100 iterations. At least 10,000 histories were used for each calculation.*

spectral radius approaches 1, the number of histories required becomes significant and effectively impractical to compute. Even with this simple diffusion problem, the behavior is consistent with that observed for the fuel assembly problem with SP_1 discretization. In that case, we estimated a spectral radius of ≈ 0.998, larger than any of the spectral radii that could be computed within even 90 minutes of compute time for this simple two dimensional problem. For that problem, even 1,000,000 histories (≈ 50 per DOF) was not enough to provide convergence. Even if single solve times of an hour can be tolerated, dozens of solves are typically required to compute the multiplication factor and flux spectrum within the SP_N eigenvalue scheme for more difficult problems like the fuel assembly, making the method unusable.

In addition to the significantly larger number of histories required to achieve convergence for ill-conditioned problems another penalty is paid due to histories that take longer to compute. Figure 3.12 gives the CPU time in seconds required to

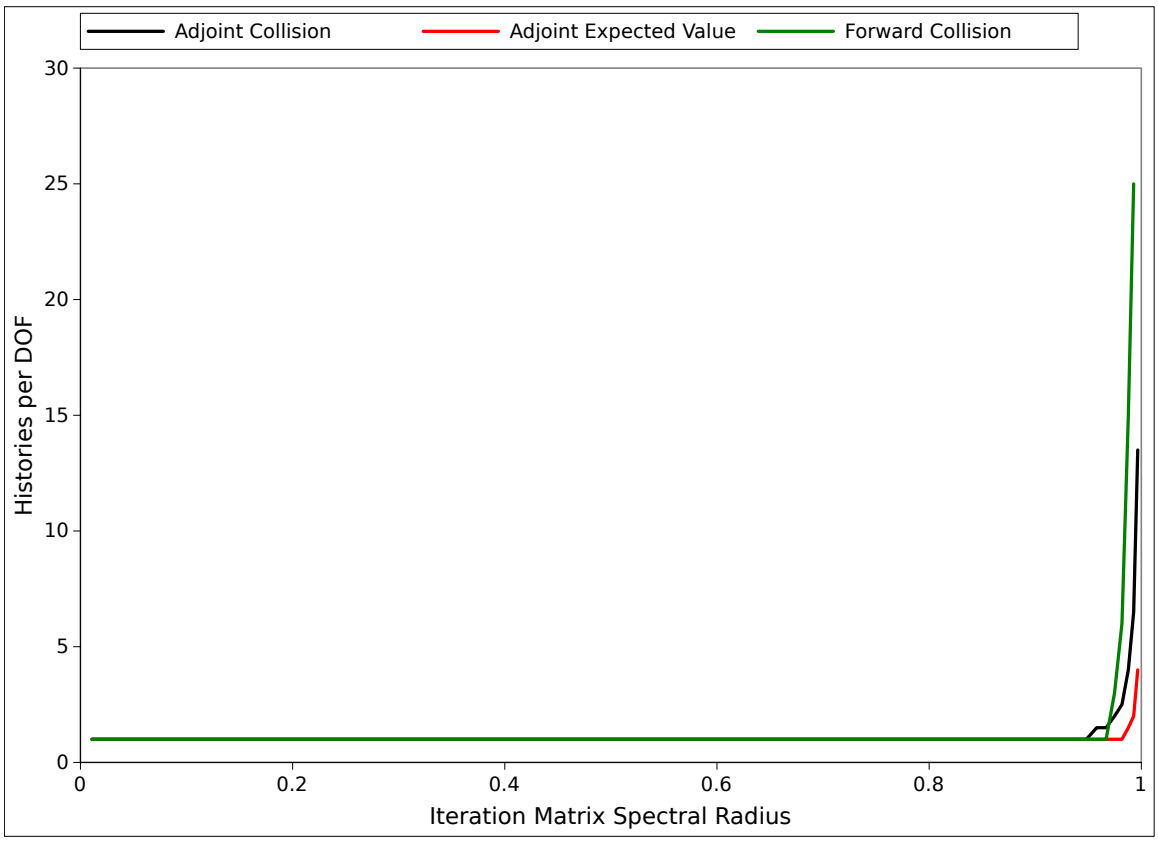

Figure 3.11: **Histories per DOF required to converge as a function of spectral radius for the neutron diffusion problem.** *The number of histories was increased to achieve convergence in less than 100 iterations. At least 10,000 histories were used for each calculation with 10,000 DOFs in the problem.*

compute a single random walk averaged over the entire set of histories run in the calculation over all iterations. As the spectral radius increases (correlating to a higher ratio of scattering in the system) the random walk lengths increase, using more CPU time to finish the computation. Compared to spectral radii of 0.5, larger spectral radii over 0.97 have histories that require two orders of magnitude more computation time. This significant increase in computation time per history coupled with the significant increase in the number of histories required to converge is evidence that for problems with spectral radii above ≈ 0.97, using MCSA to solve any problems of interest is entirely ineffective and not practical. We therefore require a more expansive set of preconditioning techniques to move the eigenvalue spectrum of the SP_N problem into a regime in which MCSA is more applicable and in which performance is improved.

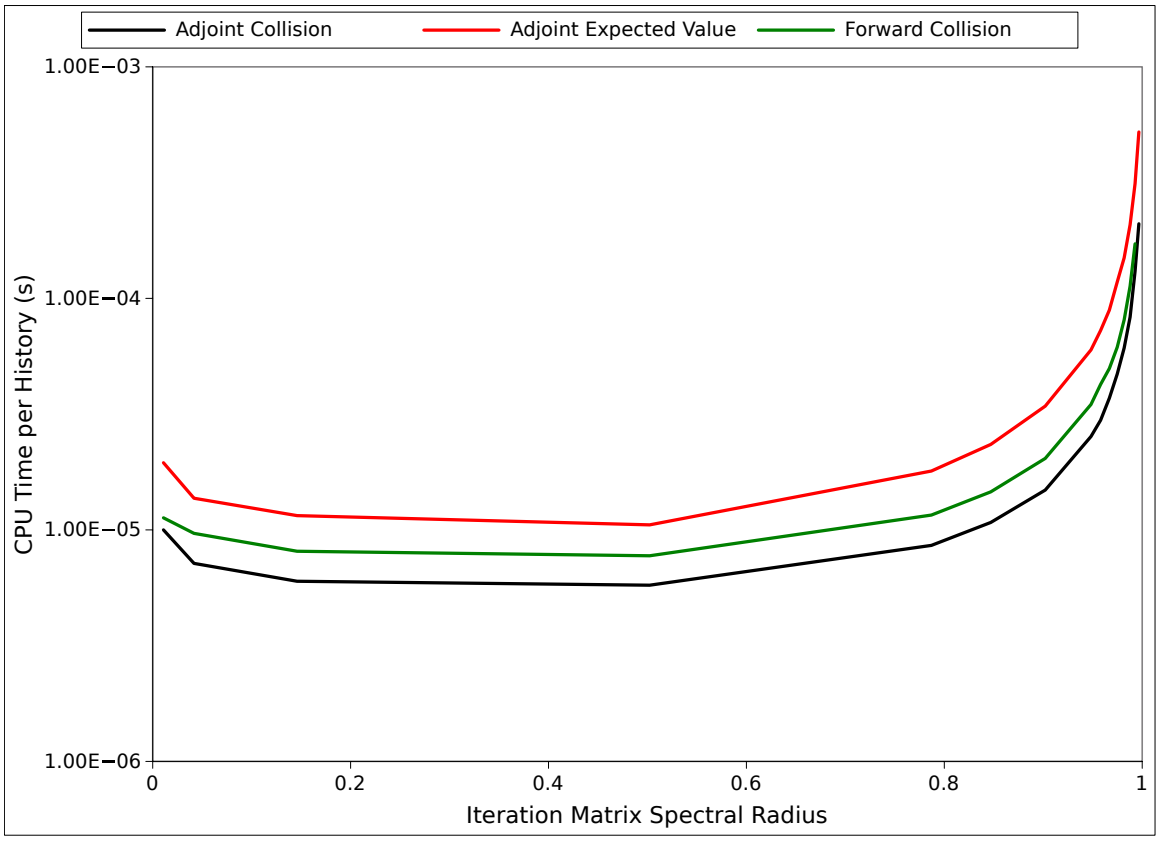

Figure 3.12: **CPU time per history as a function of spectral radius for the neutron diffusion problem.** *As the spectral radius grows, so do the length of the random walks. Longer random walks require more CPU time to compute.*

3.5 Advanced Preconditioning Strategies

For the fuel assembly criticality problem presented in the previous section, the spectral radius was near one using Jacobi preconditioning and in a regime in which MCSA breakdown is observed. In this regime the number of stochastic histories required to converge MCSA increases rapidly and the resulting poor performance is compounded by those histories being increasingly expensive to compute. To overcome this difficulty, advanced preconditioning strategies for the SP_N problem are required beyond simple Jacobi methods that can reduce the spectral radius into a region of better MCSA performance. Several modern algebraic preconditioning strategies will be presented here and used within the explicit preconditioning framework given by Eq (2.74). Data showing their effects on MCSA solutions of the fuel assembly criticality problem will be presented.

3.5.1 ILUT Preconditioning

Incomplete lower-upper (ILU) factorizations of the linear operator can be used a simple mechanism to form an approximate inverse of a preconditioner. To build the factorization, the sparse upper and lower triangular factors, $\mathbf{L}$ and $\mathbf{U}$, are computed such that the residual matrix formed by the factorization (Saad, 2003):

$$\mathbf{R} = \mathbf{LU} - \mathbf{A}\,, \tag{3.18}$$

has a specified sparsity pattern and element magnitude. When a magnitude threshold is used, elements generated in the factors below that magnitude are dropped, resulting in ILU threshold (ILUT) preconditioning. The sparsity pattern in this case is determined from the input matrix to be preconditioned and the number of elements maintained in the factor is specified by a fill level parameter. A fill level of 1 will generate the same number of elements as the sparsity pattern of the input matrix while a fill level of 2 will contain twice as many elements in the factorization, resulting in a better representation of the true LU factorization. The inverse of the lower and upper triangular factors may then be easily inverted by means of simple elimination to produce the preconditioner. For MCSA, $\mathbf{L}^{-1}$ will be used on the left and $\mathbf{U}^{-1}$ on the right to precondition the system.

For modern subspace methods, only the action to the preconditioner on a vector is required for efficient implementations and therefore a triangular solve can be used for this purpose. For the explicit preconditioning scheme presented in Eq (2.74), the

fully inverted operator must be generated in order to build the set of probabilities and weights required for Monte Carlo sampling. Therefore, for a linear system of size N, N triangular solves will be required in order to extract the inverse matrices from production ILU implementations. In addition, parallel implementations typically generate lower and upper factors that are only triangular locally, providing an easily parallelizable mechanism to generate the action of the preconditioner inverse. A consequence of this choice for parallel scalability is a degradation of the preconditioning quality as the size of the parallel system is increased. For serial computations, the triangular factorization is potentially exact depending on the parameters chosen while at thousands of parallel tasks, the global triangular factors differ significantly from the true factorization. As a result, more iterations are required at higher levels of parallelism to converge the system and will ultimately degrade overall scalability of the system with respect to total wall time to converge.

MCSA preconditioned with ILUT was used to solve the fuel assembly problem presented in the previous section for a 1-group SP_1 discretization. Unlike the Jacobi preconditioned strategy, convergence was achieved with ILUT preconditioning. To study convergence sensitivity to ILUT parameters, the fill level and drop tolerance were parametrically varied with the number of iterations to converge an eigenvalue iteration for the fuel assembly problem recorded along with the maximum number of non-zero entries observed in all matrix rows for the left/right preconditioned composite linear operator. To provide some sparsity to the factorization, the ILUT drop tolerance was used to drop elements in the extracted inverse triangular factors. For each calculation, the number of iterations required to converge reported was for a single eigenvalue iteration with 3×10^4 histories at each MCSA iteration to compute the Monte Carlo correction using the adjoint collision estimator. All ILUT calculations reported here were performed on a single CPU and therefore this data does not take into account the aforementioned effects of parallel decomposition on the quality of the ILUT preconditioning.

Figure 3.13 gives the number of iterations to converge the fixed source problem to a tolerance of 1×10^8. As expected, the higher the fill level chosen for the ILUT factorization, the fewer iterations are required to converge the problem (only 9 MCSA iterations were required to converge the problem for the fill level of 3 and drop tolerance of 1×10^{-5} case). For higher levels of fill, the iterations needed to converge were not as sensitive, signaling that a larger drop tolerance can perhaps be used without a significant degradation in iterative performance. At smaller fill levels, the sensitivity to the ILUT drop tolerance is more significant. For all fill levels used, convergence was

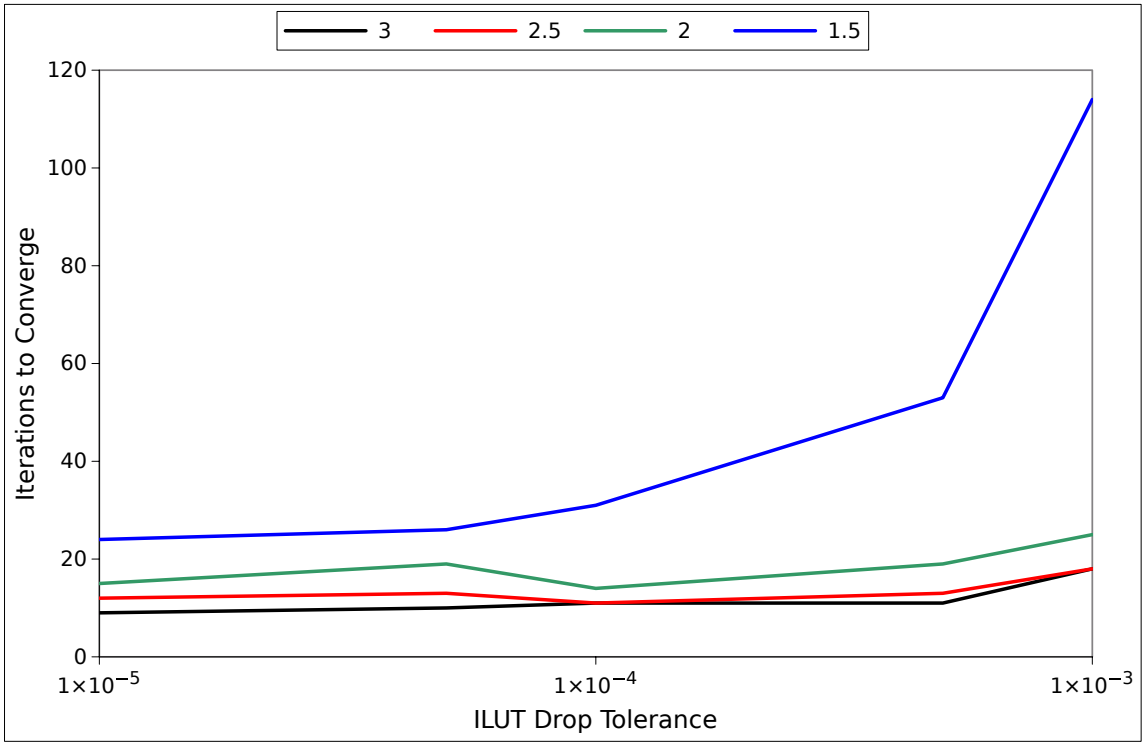

Figure 3.13: **Number of MCSA iterations required to converge an eigenvalue iteration for the fuel assembly problem with ILUT preconditioning as a function of ILUT drop tolerance.** *Each colored curve represents the iteration behavior for a different ILUT fill level. Fill levels of 1.5, 2.0, 2.5, and 3.0 were used.*

not achieved for the fuel assembly problem with a drop tolerance larger than 1×10^{-3}.

Unfortunately, gaining convergence (and excellent iterative performance) with MCSA for the SP_N fuel assembly problem comes at an immediate cost. Figure 3.14 gives the maximum number of non-zero entries in the composite linear operator generated by preconditioning as a function ILUT drop tolerance for varying values of fill level. For the 1-group SP_1 discretization, the original linear operator will contain only a maximum of 7 non-zero entries per row in the system. As observed in Figure 3.14, performing the explicit preconditioning yields composite linear operators with $O(1,000)$ elements in a row for all combinations of fill level and drop tolerance, over 10% of the total row size for this particular problem. This large number of row entries, observed for a significant fraction of rows in the system, creates several problems. First, sparsity is completely destroyed with each state in the system now coupled to over 10% of the total states in the system through the composite

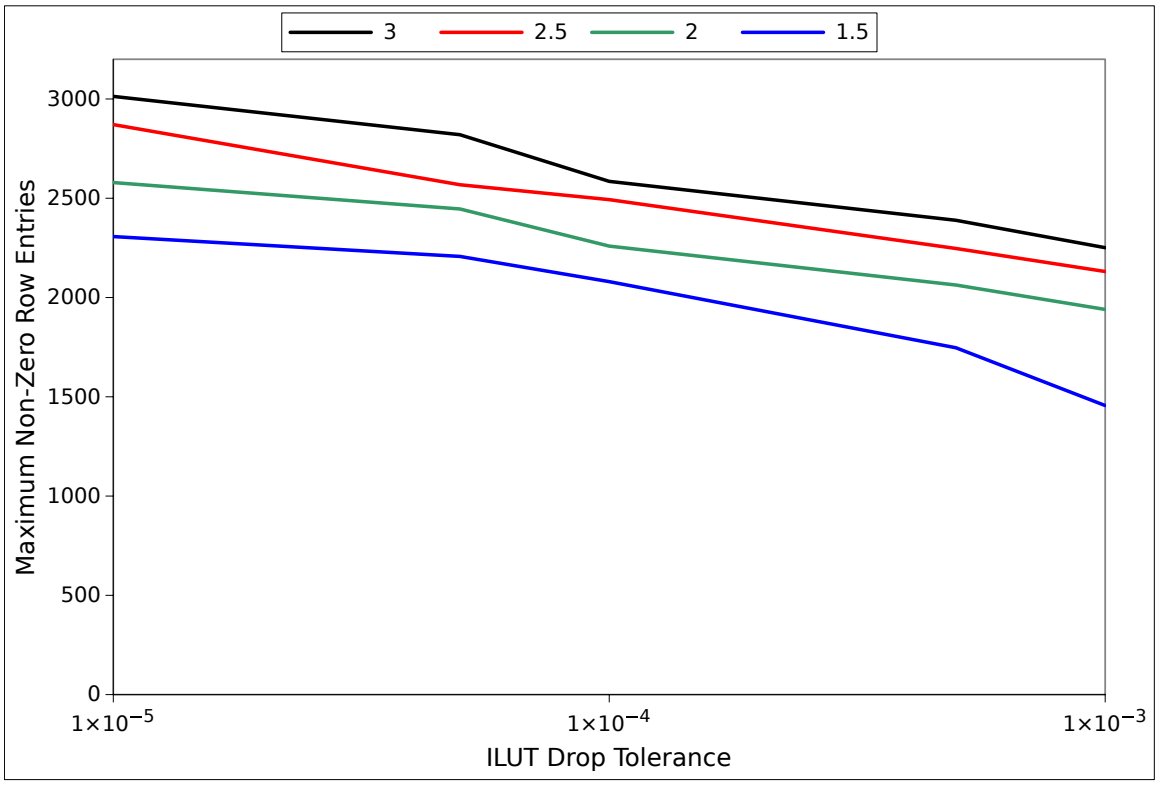

Figure 3.14: **Maximum number of non-zero entries observed for all rows in the composite linear operator for the fuel assembly problem with ILUT preconditioning given as a function of ILUT drop tolerance.** *Each colored curve represents the row size for a different ILUT fill level. Fill levels of 1.5, 2.0, 2.5, and 3.0 were used.*

iteration matrix. This means that Monte Carlo sampling tables will be large, requiring significant memory to store them and a substantial overhead in the sampling procedure during transport. Second, the matrix-matrix multiply operations required to build the composite operator see significant performance losses due to the amount of data that must be handled. In parallel, losing sparsity severely inhibits performance where now parallel operations require communications amongst orders of magnitude more processors for nearest-neighbor type algorithms.

As a means to assess the quality of the preconditioning, a simple metric is developed to address these concerns and allow comparison to future developments. For our studies, our core performance metric will be iterative performance with the minimum number of MCSA iterations required to converge the problem desired. For Monte Carlo calculations, this improved performance is balanced by the creation of a dense composite system and we seek to reduce the number of non-zero entries to a minimum

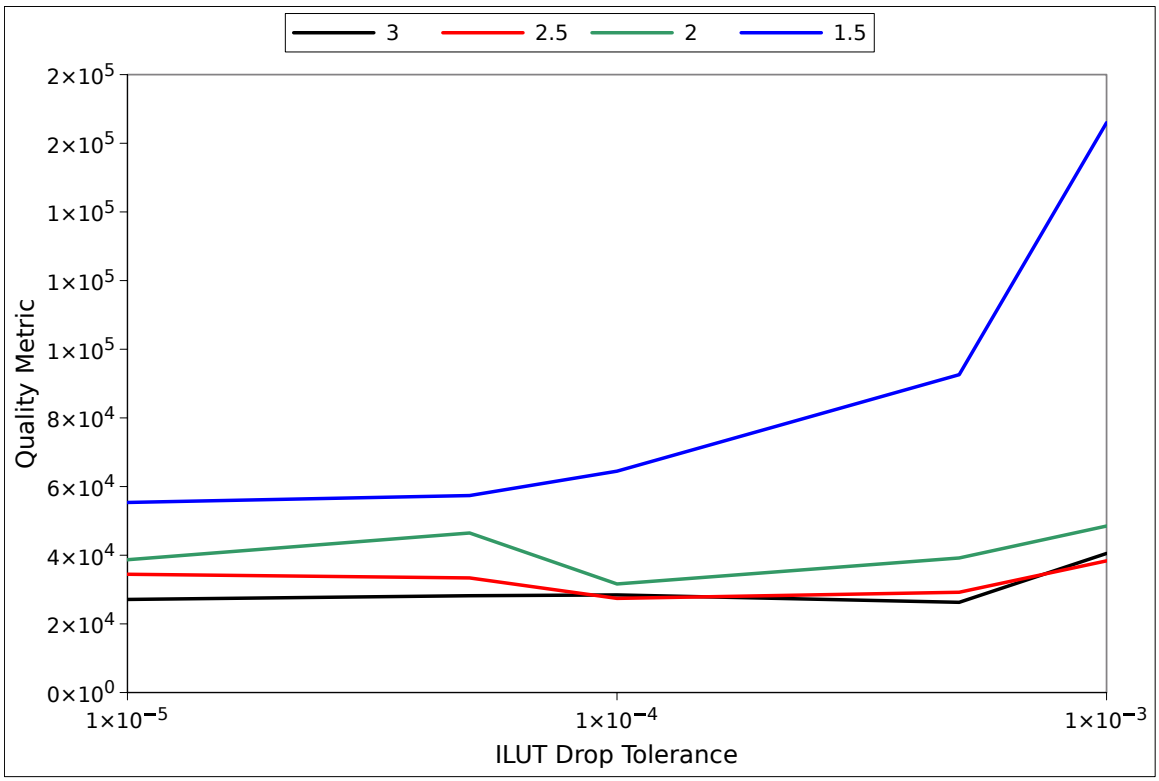

Figure 3.15: **ILUT preconditioning quality metric for the fuel assembly problem given as a function of ILUT drop tolerance.** *Each colored curve represents the quality metric behavior for a different ILUT fill level. Fill levels of 1.5, 2.0, 2.5, and 3.0 were used.*

value in order to lower the amount of coupling amongst states in the system and reduce the amount of memory used along with the potential latency overhead. Given these two objectives, the following metric is proposed:

$$\text{Quality Metric} = (\#\text{ iterations}) \times (\text{maximum } \#\text{ of non-zero values}) \,, \tag{3.19}$$

where the highest quality preconditioning is one that minimizes this metric. For the ILUT preconditioned data provided, Figure 3.15 provides the computed metric as a function of ILUT drop tolerance for each fill level used. In general, the metric is similar to the data observed for the number of iterations required to converge as the non-zero row entries changed by a smaller fraction over drop tolerances tested.

It should be noted here that although only the behavior of the linear solve during a single eigenvalue iteration is reported, the behavior of the linear solver was observed to be consistent throughout the eigenvalue iterations (25 total eigenvalue iterations

were required to converge the 1-group SP_1 problem). We expect this as the linear operator (which is unchanging in the fuel assembly criticality problem) dictates the convergence of MCSA. At each iteration the fission source provided to the linear problem is changing, however, we expect the same convergence behavior for all source vectors $\mathbf{b} \in \mathbb{R}^N$ where $||\mathbf{b}||_1 \neq 0$.

3.5.2 Sparse Approximate Inverse Preconditioning

Explicit formation of the preconditioner inverse is a requirement of the current MCSA preconditioning strategy. As a result, although ILUT preconditioning provided excellent iterative performance, a significant time penalty is paid to extract the inverse of the triangular factors through N triangular solves. In addition, this explicit inversion yielded a composite linear operator that was dense, requiring modification of the system in order to achieve better scalability and Monte Carlo performance. Sparse approximate inverse (SPAINV) preconditioning is a technique that may potentially alleviate both of these constraints at the cost of reduced iterative performance. The method produces a sparse approximation of the inverse matrix directly by minimizing the Frobenius norm of the residual matrix (Saad, 2003):

$$||\mathbf{I} - \mathbf{AM}||_F^2 = \sum_{j=1}^{N} ||\mathbf{e}_j - \mathbf{Am}_j||_2^2 \,, \tag{3.20}$$

where the inverse preconditioning matrix $\mathbf{M}$ minimizes the norm on the right and $\mathbf{e}_j$ and $\mathbf{m}_j$ are the j^{th} columns of the identity and preconditioning matrices respectively. Here, $\mathbf{M}$ is the actual inverse of the preconditioner and is formed directly by the algorithm. On the right hand side of Eq (3.20), the Frobenius norm is represented as a effective sum of N linear system residual norms.

A few iterations of a subspace method or other iterative scheme can be used to effectively solve these systems to a loose tolerance and yield the columns of the approximate preconditioning matrix. At each iteration, threshold values are used to eliminate the components of each $\mathbf{m}_j$ column to maintain the desired level of sparsity. In addition, a sparsity pattern can be predetermined and enforced during this process. For the SPAINV implementation used for this work, the sparsity pattern was defined by levels such that a level of N for an input operator $\mathbf{A}$ will generate a preconditioning matrix $\mathbf{M}$ with the same sparsity pattern as $\mathbf{A}^{N+1}$. By using the norm minimization strategy instead of a factorization such as ILUT, parallel results are reproduced regardless of parallel problem size, meaning that a serial computation

will converge in the same number of iterations as a computation with thousands of cores.

MCSA preconditioned with SPAINV was used to solve the fuel assembly problem as with ILUT for a 1-group SP_1 discretization. To study convergence sensitivity to SPAINV parameters, the number of levels in the sparsity pattern and threshold were parametrically varied with the number of iterations to converge a single eigenvalue iteration for the fuel assembly problem recorded along with the maximum number of non-zero entries observed in all matrix rows for the left/right preconditioned composite linear operator. For each calculation, the number of iterations required to converge reported was for a single eigenvalue iteration with 3×10^4 histories at each MCSA iteration to compute the Monte Carlo correction using the adjoint collision estimator. All SPAINV calculations reported here were performed on a single CPU.

Figure 3.16 gives the number of MCSA iterations needed to converge the fuel assembly problem and Figure 3.17 the maximum number of non-zero entries per row observed in the composite linear operator as a function the SPAINV threshold. For this analysis, the number of levels in the sparsity pattern was varied from 3 to 7 with convergence of the fuel assembly problem not achieved for smaller level values. Compared to ILUT preconditioning, at higher levels we see comparable iterative performance with a relative invariance to the threshold value. The threshold did have a significant effect, however, on the time required to generate the preconditioner with a threshold of 0.001 over an order of magnitude slower than a threshold of 0.01 due to the inclusion of a significant number of extra values in the input operator. In addition to comparable iterative performance, the sparsity of the composite operator is greatly improved with nearly an order of magnitude reduction in number of non-zero entries per row for lower level values.

With the comparable iterative performance and improved sparsity, SPAINV preconditioning should then have a more favorable quality metric than ILUT preconditioning. Figure 3.18 gives the quality metric as a function of the threshold value for each of the sparsity levels computed. Not only do we see an improved quality metric over ILUT preconditioning (about an order of magnitude less), but we also note that the ideal sparsity level is not the largest nor the smallest. In fact, the largest and smallest levels (3 and 7) performed the worst in terms of the quality metric with a level of 4 performing the best. When CPU time to generate the preconditioner is considered, 4 is the clear choice for this problem as more time is required to do the approximate inversion as the sparsity pattern of the preconditioner grows in size.

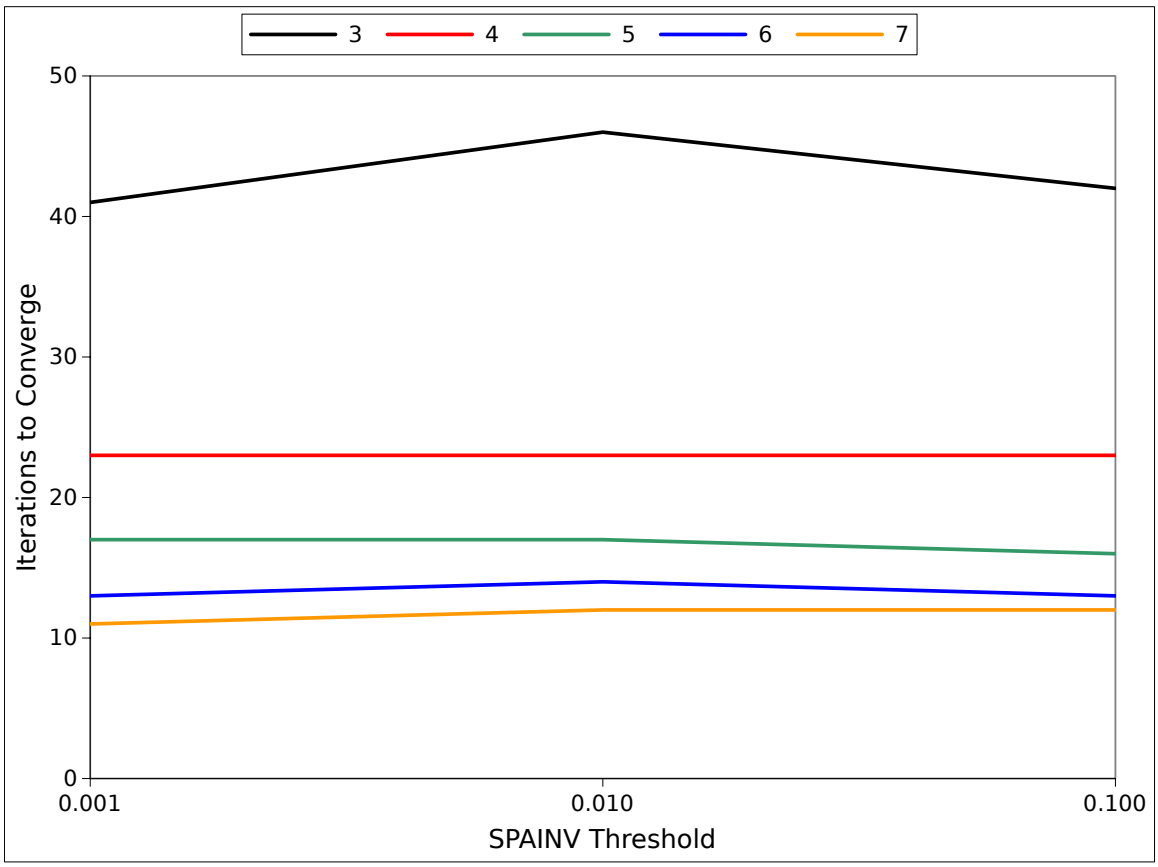

Figure 3.16: **Number of MCSA iterations required to converge a single eigenvalue iteration for the fuel assembly problem with SPAINV preconditioning as a function of SPAINV threshold.** *Each colored curve represents the iteration behavior for a different SPAINV level pattern. Levels of 3, 4, 5, 6, and 7 were used.*

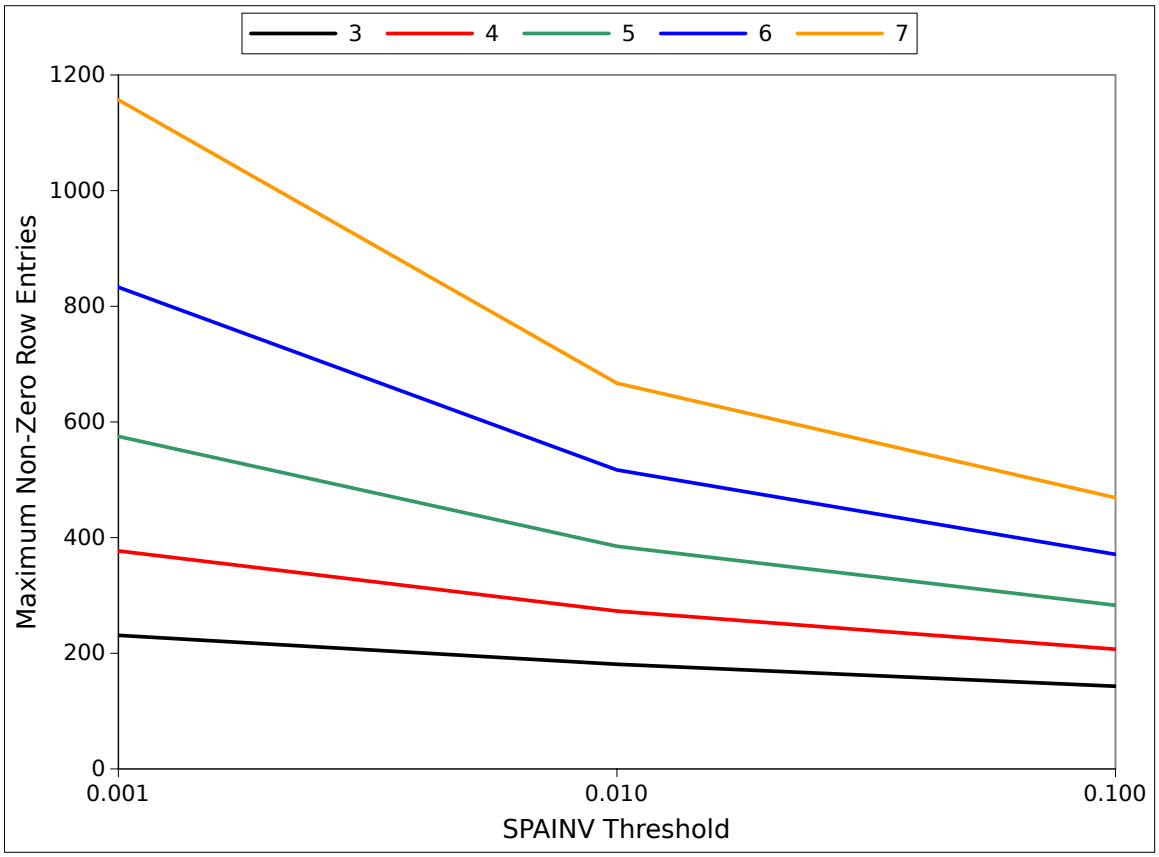

Figure 3.17: **Maximum number of non-zero entries observed for all rows in the composite linear operator for the fuel assembly problem with SPAINV preconditioning given as a function of SPAINV threshold.** *Each colored curve represents the row size for a different SPAINV level pattern. Levels of 3, 4, 5, 6, and 7 were used.*

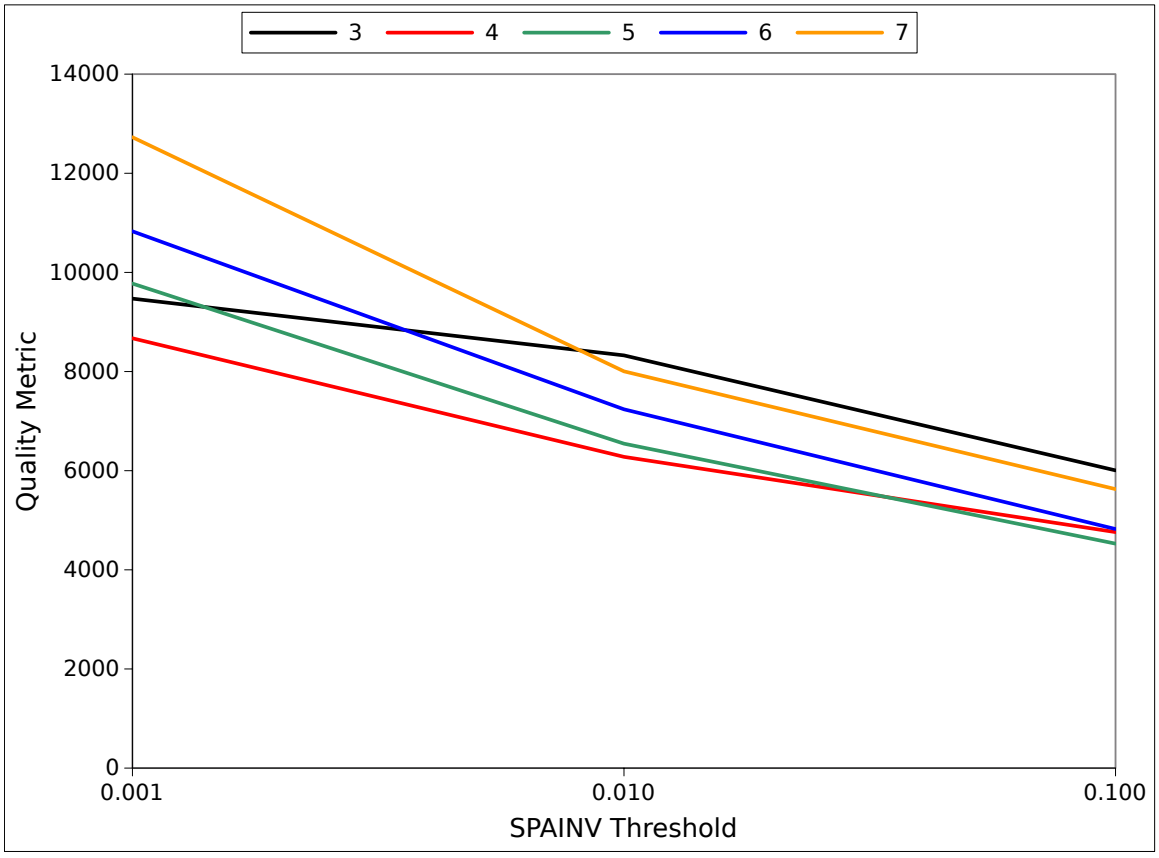

Figure 3.18: **SPAINV preconditioning quality metric for the fuel assembly problem given as a function of SPAINV threshold.** *Each colored curve represents the quality metric behavior for a different SPAINV level pattern. Levels of 3, 4, 5, 6, and 7 were used.*

3.5.3 SPAINV Improved ILUT Preconditioning

As a means to improve convergence over using only SPAINV preconditioning, ILUT preconditioning may be modified through an additional application of SPAINV (SPAINV-ILUT)(Saad, 2003). For the ILUT preconditioning case we have the following linear problem to solve for the substituted variable:

$$\mathbf{L}^{-1}\mathbf{A}\mathbf{U}^{-1}\mathbf{u} = \mathbf{L}^{-1}\mathbf{b}\,. \tag{3.21}$$

We then apply SPAINV and minimize the following problem, this time on the left:

$$||\mathbf{I} - \mathbf{M}\mathbf{L}^{-1}\mathbf{A}\mathbf{U}^{-1}||_F^2\,, \tag{3.22}$$

giving a new preconditioned system:

$$\mathbf{M}\mathbf{L}^{-1}\mathbf{A}\mathbf{U}^{-1}\mathbf{u} = \mathbf{M}\mathbf{L}^{-1}\mathbf{b}\,. \tag{3.23}$$

Doing this permits a smaller fill level to be used with ILUT and a lower-order sparsity pattern to be used with SPAINV, giving marginally more sparsity in the resulting composite linear operator while maintaining good iterative performance.

MCSA preconditioned with SPAINV-ILUT was also used to solve the same fuel assembly problem as with the other preconditioners for the same 1-group SP_1 discretization. To study convergence sensitivity to SPAINV-ILUT parameters, the ILUT drop tolerance and ILUT fill level were parametrically varied with the number of iterations to converge a single eigenvalue iteration for the fuel assembly problem recorded along with the maximum number of non-zero entries observed in all matrix rows for the left/right preconditioned composite linear operator. SPAINV parameters for the data presented here were fixed as it was found that the preconditioning quality was largely insensitive to their variation. In addition, the fact that SPAINV is being used to preconditioned a problem already preconditioned with ILUT means that the input matrix sparsity pattern is already somewhat dense and large SPAINV levels would result in an even denser system. Therefore, a smaller SPAINV sparsity pattern level will be used for this preconditioning. For each calculation, the number of iterations required to converge reported was for a single eigenvalue iteration. Again, 3×10^4 histories are used at each MCSA iteration to compute the Monte Carlo correction using the adjoint collision estimator. All SPAINV-ILUT calculations reported here were performed on a single CPU.

Figure 3.19 gives the number of iterations required to converge as a function of ILUT drop tolerance for ILUT fill levels of 1.5, 2.0, 3.0, 4.0 and 5.0 with a fixed SPAINV threshold of 1.0 and sparsity level of 1. SPAINV threshold values higher than 1.0 resulted in a loss of convergence and lower values did not significantly affect performance. Larger sparsity levels resulted in a composite operator that was too dense and a prohibitive amount of CPU time required to compute the preconditioner. At smaller levels of fill, the benefits of using an additional step of SPAINV preconditioning are readily observed with the number of iterations required to converge cut in half for large drop tolerances. For smaller drop tolerances and higher ILUT levels of fill, the additional step of SPAINV preconditioning offers marginal improvement to iterative performance. As shown in Figure 3.20, because ILUT is used in the preconditioning sequence, sparsity is again lost with only marginally improved non-zero entry numbers due to the application of the SPAINV threshold. As a result, the quality metrics given by Figure 3.21 are only marginally better than those observed for ILUT preconditioning and much larger than the SPAINV quality metric values.

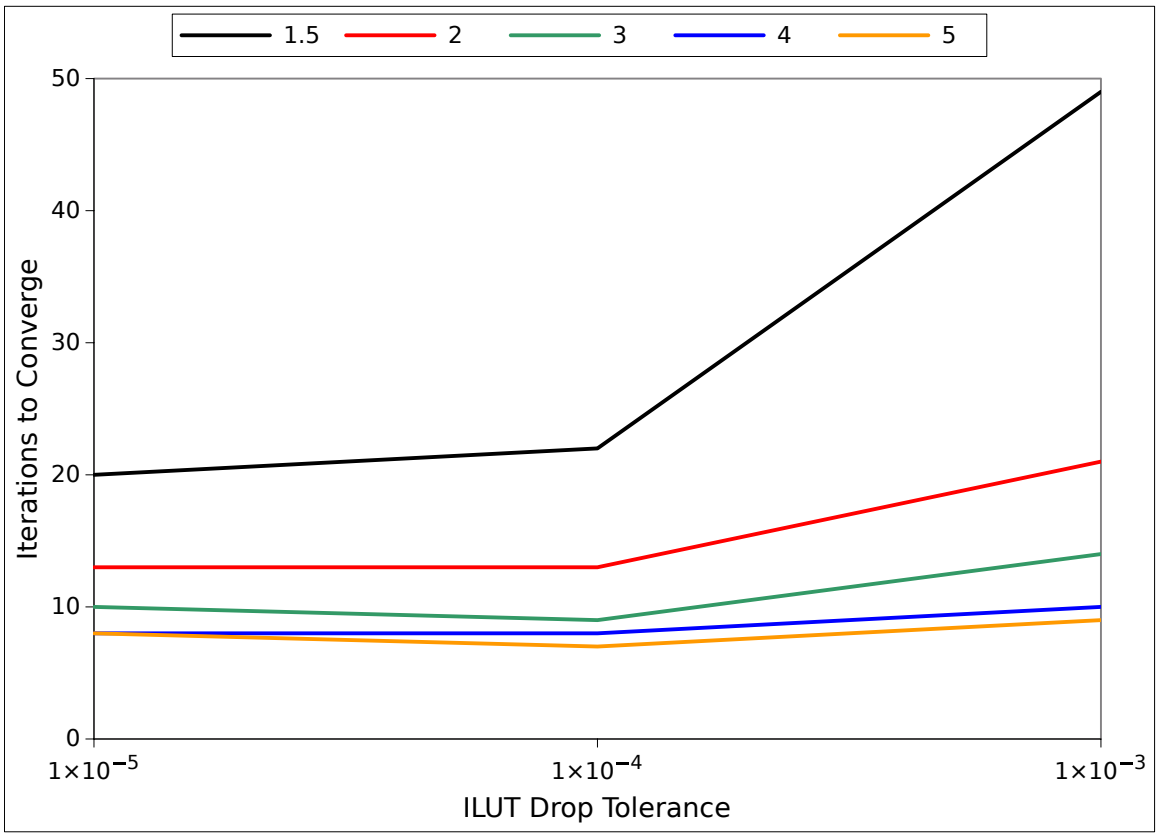

Figure 3.19: **Number of MCSA iterations required to converge a single eigenvalue iteration for the fuel assembly problem with SPAINV-ILUT preconditioning as a function of ILUT drop tolerance.** *Each colored curve represents the row size for a different ILUT fill level. Fill levels of 1.5, 2.0, 3.0, 4.0, and 5.0 were used.*

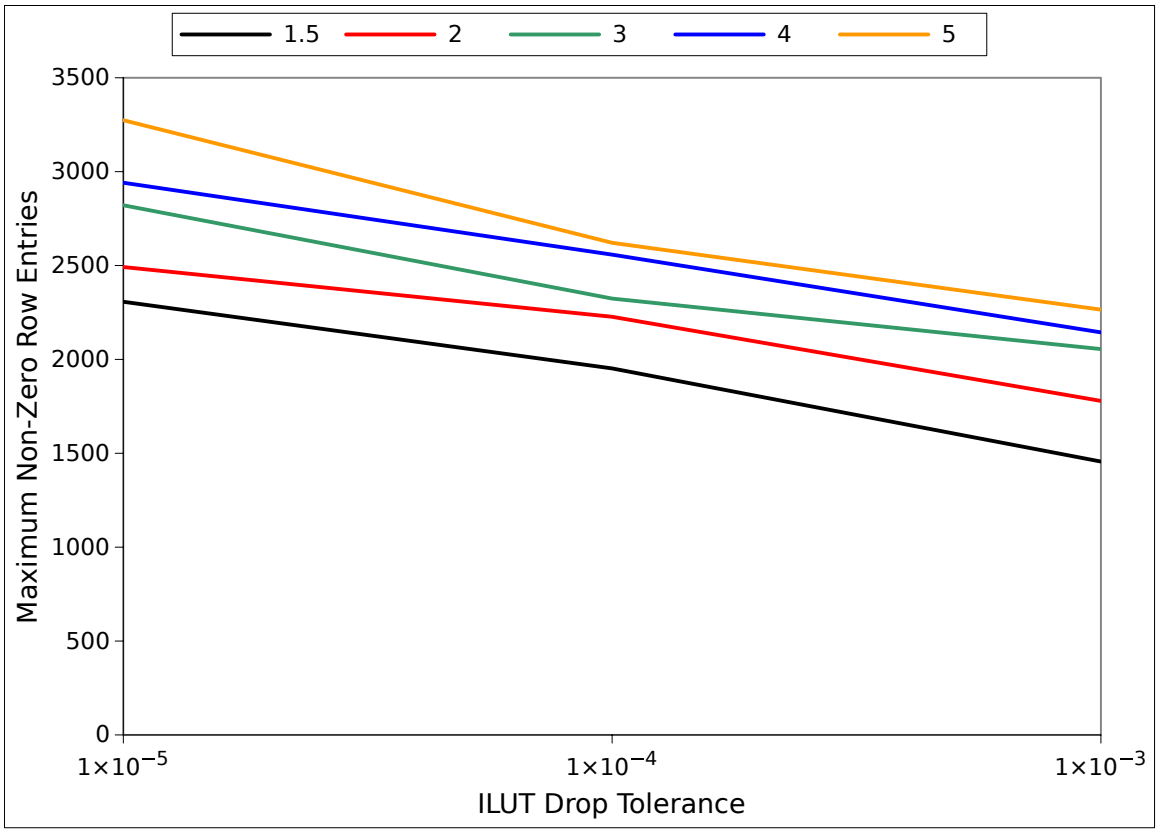

Figure 3.20: **Maximum number of non-zero entries observed for all rows in the composite linear operator for the fuel assembly problem with SPAINV-ILUT preconditioning given as a function of ILUT drop tolerance.** *Each colored curve represents the row size for a different ILUT fill level. Fill levels of 1.5, 2.0, 3.0, 4.0 and 5.0 were used.*

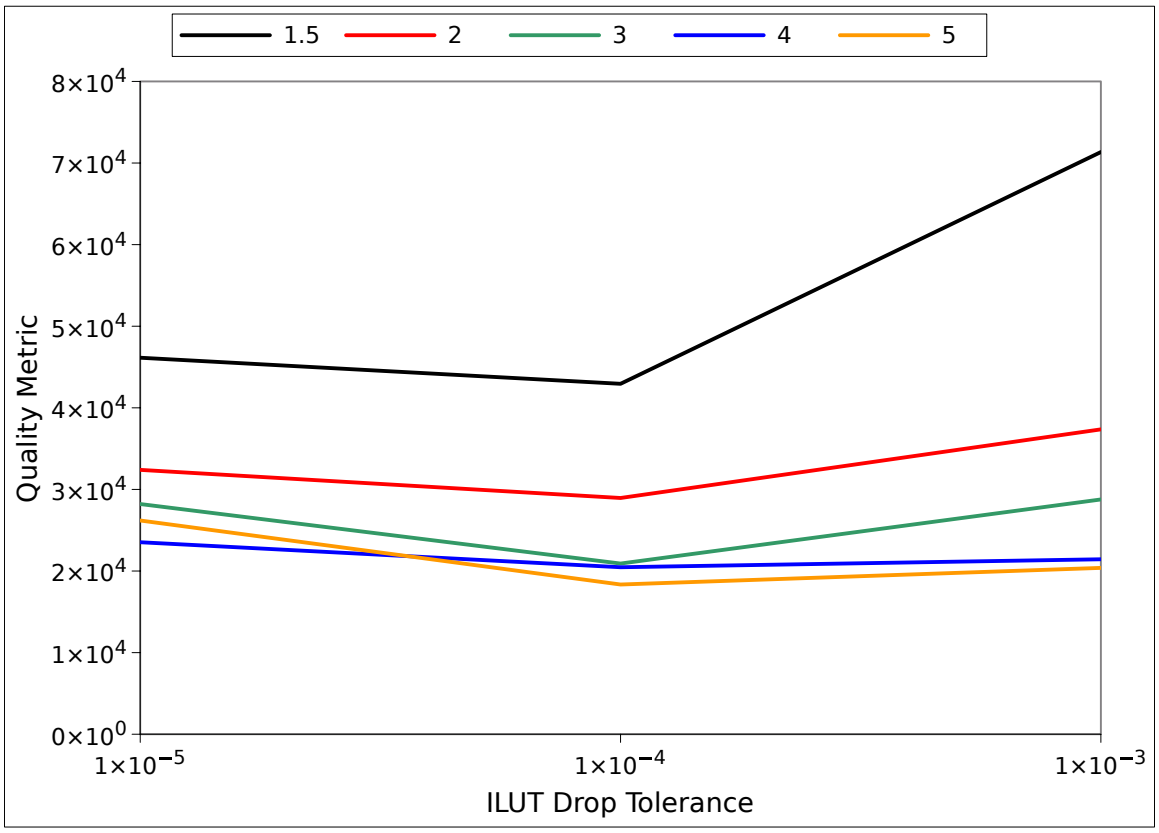

Figure 3.21: **SPAINV-ILUT preconditioning quality metric for the fuel assembly problem given as a function of ILUT drop tolerance.** *Each colored curve represents the row size for a different ILUT fill level. Fill levels of 1.5, 2.0, 3.0, 4.0 and 5.0 were used.*

3.5.4 Applying the Reduced Domain Approximation

For each preconditioning technique presented convergence was achieved for the single fuel assembly problem. However, a primary concern is the number of non-zero states in each row of the system generated by the explicit preconditioning strategy. In many cases, orders of magnitude more matrix elements were generated resulting in poor scalability for domain decomposed algorithms and overall performance issues for Monte Carlo. As outlined in § 2.10.2, the reduced domain approximation may be used as a mechanism to potentially alleviate this problem by filtering elements of the composite matrix in each row that fall below a certain threshold value or by maintaining the largest N elements in each row where N is a designated fill level.

For the ILUT, SPAINV, and SPAINV-ILUT preconditioning strategies the reduced domain approximation will be applied to reduce the density of the composite linear operator to more manageable levels. For each preconditioner, the parameters that achieved the best quality metric results from the previous analysis were used. These correspond to ILUT parameters of a fill level of 5.0 and a drop tolerance of 1×10^{-5}, SPAINV parameters of a sparsity level of 4 and a threshold of 0.1 and SPAINV-ILUT with ILUT parameters of a fill level of 4.0 and a drop tolerance of 1×10^{-5} and SPAINV parameters of a sparsity level of 1 and a threshold of 1.0. The reduced domain threshold was set to 1×10^{-10} in order to eliminate any exceedingly small values from the matrix (generally this is simply removing non-zero elements within the floating point tolerance of zero). The reduced domain fill level was then varied, starting with the largest non-zero entries per row value observed for each of the preconditioning types in order to assess its effects relative to the case where no reduced domain approximation was applied.

Figure 3.22 gives the number of iterations required to converge for each preconditioner type as a function of reduced domain fill level. Figure 3.23 gives the corresponding quality metric for each data point where the number of non-zero entries used to compute the metric is equivalent to the reduced domain fill level. We first note that SPAINV preconditioning alone is significantly more sensitive to the reduction in domain size over the ILUT-based methods, although the preconditioner was of $O(100)$ non-zero entries per row without any approximation applied. For the ILUT-based methods, performance was significantly better with convergence achieved in less than 40 MCSA iterations with only 10 non-zero entries in each row (vs. 7 non-zero entries for the case with no preconditioning). SPAINV-ILUT iterative performance was marginally better than ILUT alone for all reduced domain fill levels. However, it

should be noted that the ILUT level of fill used for the ILUT only calculations was set to 4 for this case while the SPAINV-ILUT preconditioner used an ILUT level of fill of 5 and therefore the marginally better performance is more likely a result of this addition of fill rather than the extra SPAINV preconditioning. Looking at the quality metric data, we see a nice power-law reduction in the quality metric as a function of the reduced domain fill level, achieving 2 orders of magnitude reduction in the quality metric for the ILUT-based preconditioning methods.

Although applying the reduced domain approximation results in a successful recovery of sparsity for the Monte Carlo problem while maintaining good convergence properties, there is still an issue of forming the composite operator before applying the approximation and potentially generating the transpose of this operator in the case of the adjoint Monte Carlo method. Because of this, memory and scaling issues may still be observed when building the probability and weight matrices for the Monte Carlo problem. In addition, the expensive extraction of the inverse of the preconditioning operators for the explicit scheme creates a significant cost in overall performance. Future work in this area should be considerate of these important components of the preconditioned MCSA algorithm.

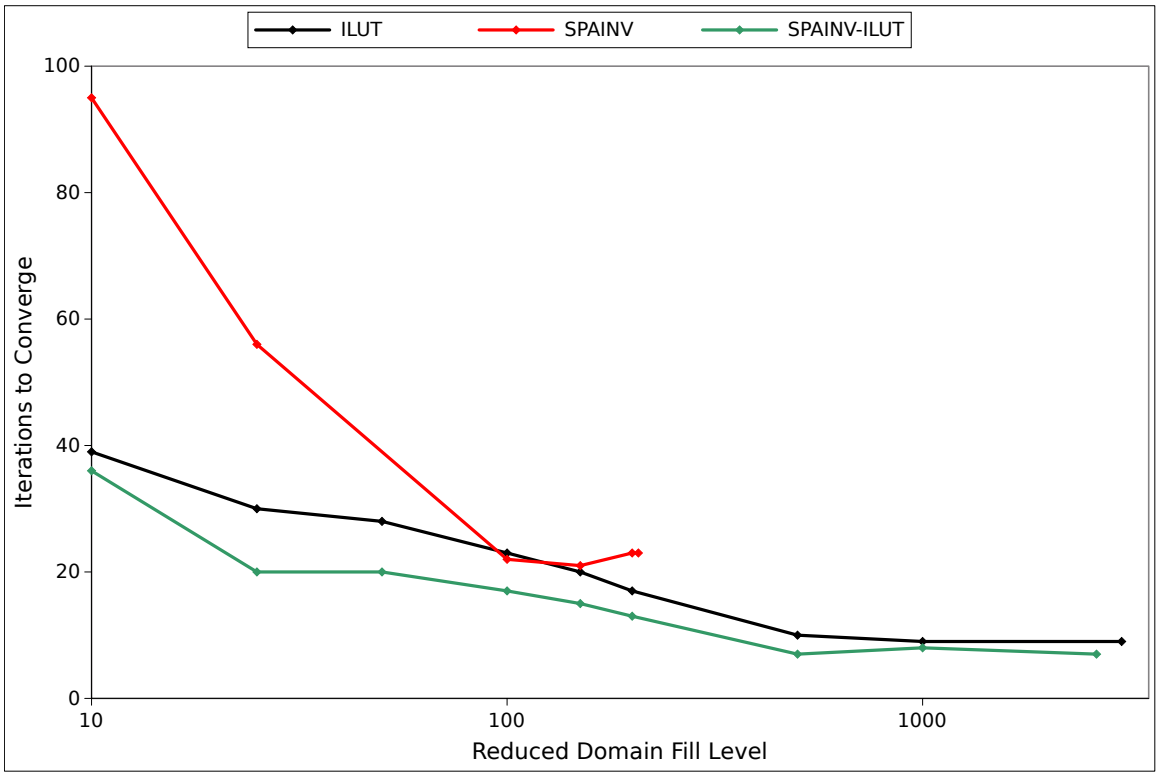

Figure 3.22: **Number of MCSA iterations required to converge a single eigenvalue iteration for the fuel assembly problem with each preconditioning as a function of reduced domain approximation fill level.** *The largest fill level for each preconditioning presented is that using the parameters that gave the best results without the approximation.*

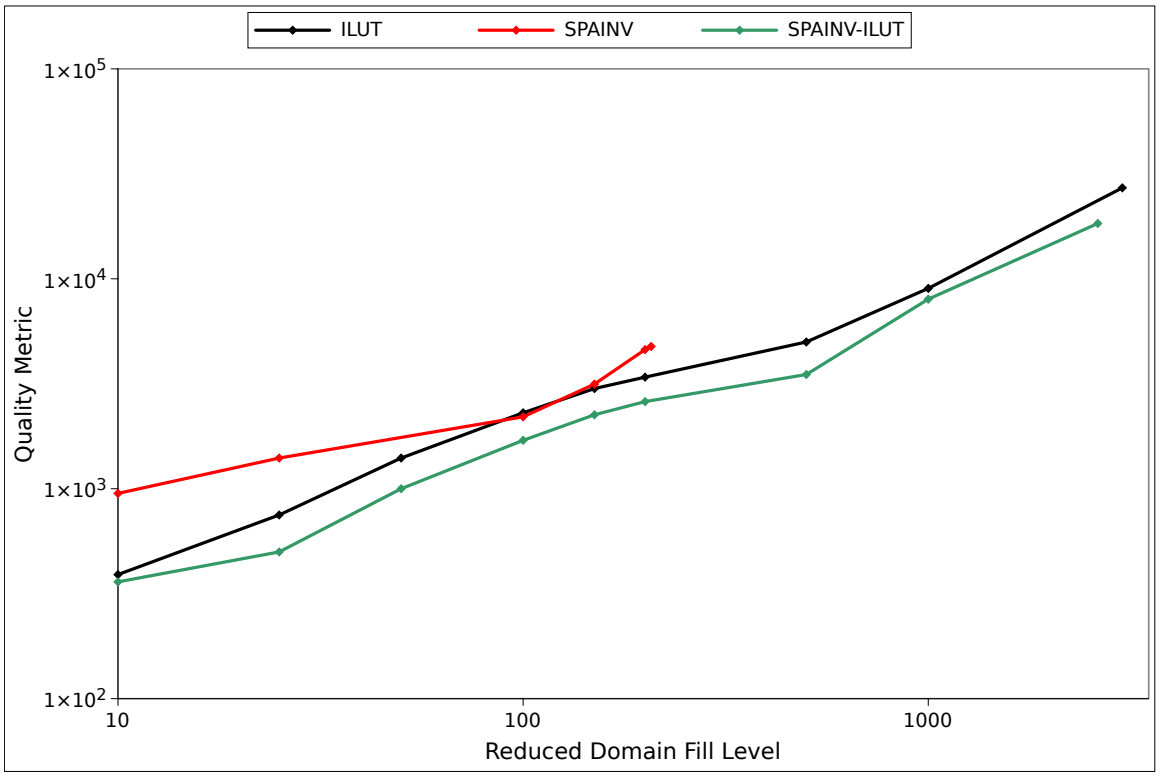

Figure 3.23: **Preconditioning quality metric for the fuel assembly problem given as a function of reduced domain approximation fill level.** *The largest fill level for each preconditioning presented is that using the parameters that gave the best results without the approximation.*

3.5.5 MCSA Relaxation Parameters

As another means of preconditioning (or variance reduction), the Richardson iteration upon which the Neumann-Ulam method is built may be implemented with a scalar relaxation parameter that can be adjusted to improve convergence:

$$\mathbf{x} = \mathbf{x} + \omega \mathbf{r} \,, \tag{3.24}$$

where ω is the relaxation parameter. This is very similar to point Jacobi preconditioning where the system is being scaled on the left by a constant value in all rows. Analogously, these relaxation parameter techniques can be applied to Monte Carlo to improve convergence as demonstrated by Dimov (Dimov et al., 1998). In this case, building an iteration matrix from Eq (3.24) gives:

$$\mathbf{H} = \mathbf{I} - \omega \mathbf{A} \,, \tag{3.25}$$

with the probabilities and weights for the Monte Carlo procedure appropriately scaled. By inspection, such a scaling is a simple form of preconditioning on the left where all rows in the system are scaled by the same scalar parameter. For MCSA, we can stage this scheme with two separate relaxation parameters; one for the outer Richardson iteration and one for the inner Monte Carlo solve:

$$\mathbf{x}^{k+1/2} = \mathbf{x}^{k} + \omega_R \mathbf{r}^{k} \,, \tag{3.26a}$$

$$\mathbf{r}^{k+1/2} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1/2} \,, \tag{3.26b}$$

$$\omega_N \mathbf{A} \delta \mathbf{x}^{k+1/2} = \omega_N \mathbf{r}^{k+1/2} \,, \tag{3.26c}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta \mathbf{x}^{k+1/2} \,, \tag{3.26d}$$

$$\mathbf{r}^{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1} \,, \tag{3.26e}$$

where ω_R is the Richardson iteration relaxation parameter and ω_N is the Neumann-Ulam Monte Carlo solve relaxation parameter.

We apply these relaxation parameters to the fuel assembly problem along with the reduced domain approximation as a means of studying their effects. For each calculation presented, the number of iterations required to converge reported was for a single eigenvalue iteration. Again, 3×10^4 histories are used at each MCSA iteration to compute the Monte Carlo correction using the adjoint collision estimator. A reduced domain fill level of 100 is used with a threshold of 1×10^{-10} to filter small values. ILUT preconditioning with a drop tolerance of 1×10^{-5} and fill level of 5 is

used as well.

Figure 3.24 gives the number of iterations required to converge the fuel assembly problem for a 1-group SP1 discretization using varying combinations of the relaxation parameters. First, both parameters were fixed at the base case of 1 while the other parameter was varied as shown in the left-hand side plots of Figure 3.24. For the Richardson relaxation parameter, using a value larger than 1 gave better iterative performance up to a point, effectively providing a stronger extrapolation using the residual at each iteration. For the Neumann relaxation parameter, a value of less than 1 is observed to be ideal. Although initially counter-intuitive, the fact that the correction computed by the Monte Carlo solver has a stochastic error associated with it means that by using a Neumann relaxation parameter less than 1, the correction and its error are effectively dampened to improve iterative performance. Considering the CPU time required to converge a single eigenvalue iteration, similar results are also observed for the relaxation parameters as given by Figure 3.25. For the base cases given by the plots on the left, it was found the a Richardson relaxation parameter of 1.1 and a Neumann relaxation parameter of 0.7 provided the fastest CPU time for convergence. Fixing each parameter at these new values, the calculations were repeated as shown for the plots on the right hand sides of both Figures 3.24 and 3.25. For each repeated timing calculation, it was found that the same combination of relaxation parameters found in the base cases performed the best although they did not necessarily have the best iterative performance.

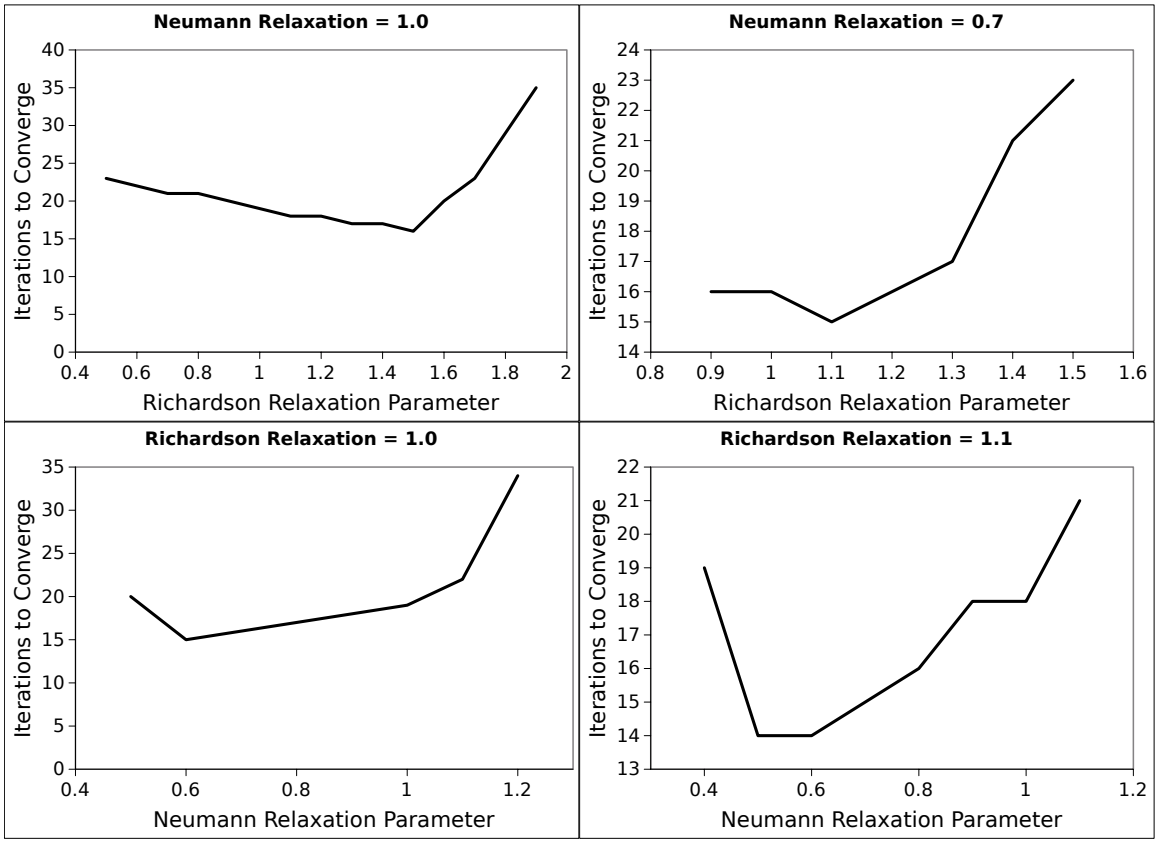

Figure 3.24: **Number of iterations to converge a single eigenvalue iteration of the fuel assembly problem as a function of the relaxation parameters.** *Starting in upper left and moving counter-clockwise: Neumann relaxation parameter fixed at 1.0, Richardson relaxation parameter fixed at 1.0, Neumann relaxation parameter fixed at 0.7, Richardson relaxation parameter fixed at 1.1. For each calculation* 3×10^4 *stochastic histories were used to compute the MCSA correction at each iteration.*

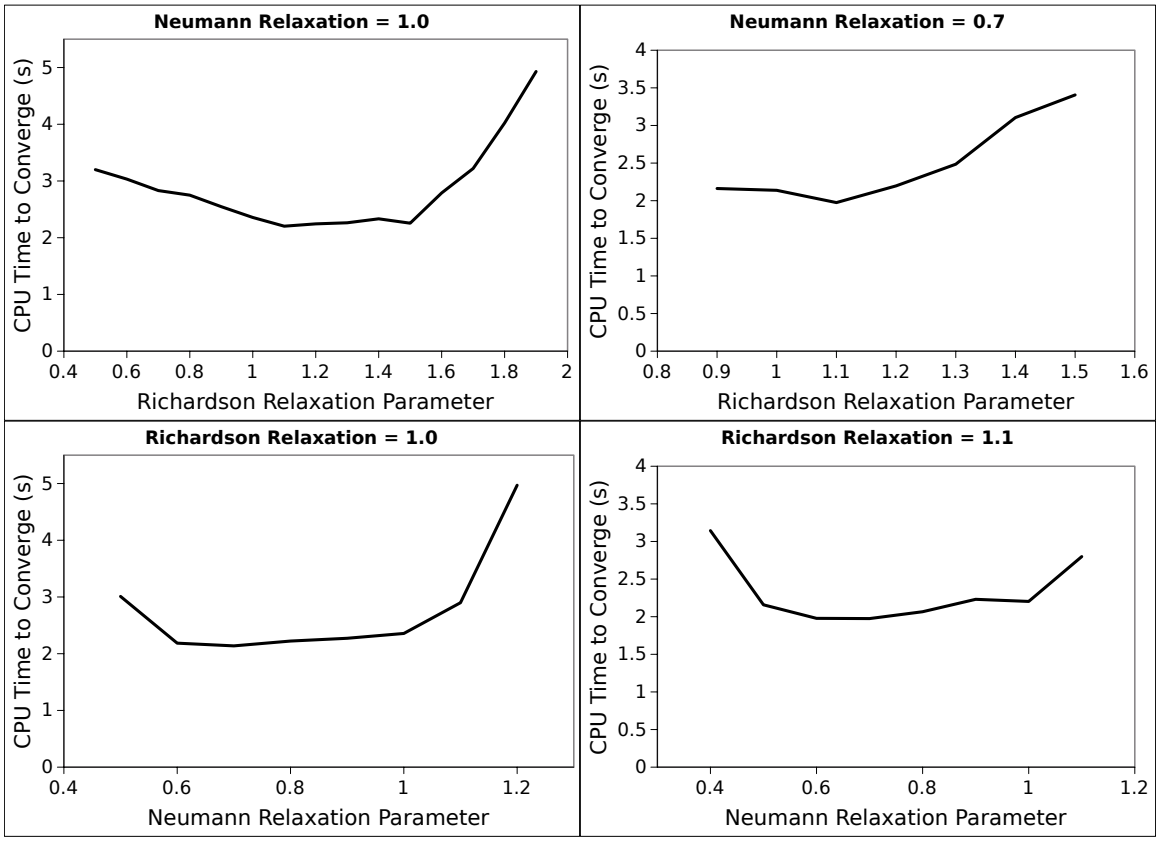

Figure 3.25: **CPU time in seconds to converge a single eigenvalue iteration of the fuel assembly problem as a function of the relaxation parameters.** *Starting in upper left and moving counter-clockwise: Neumann relaxation parameter fixed at 1.0, Richardson relaxation parameter fixed at 1.0, Neumann relaxation parameter fixed at 0.7, Richardson relaxation parameter fixed at 1.1. For each calculation* 3×10^4 *stochastic histories were used to compute the MCSA correction at each iteration.*

3.5.6 Monte Carlo Estimator Comparison

With convergence obtained for the fuel assembly problem, the advanced preconditioners will next be studied with both the collision and expected value estimators using the best preconditioning and relaxation parameter combinations found by the previous analysis. Additionally, both the Richardson-based MCSA iteration given by Eq (2.65) and the MINRES-based MCSA iteration given by Eq (2.67) will be used. Although the expected value estimator achieved the best iterative performance in the previous comparison and analysis for the simple Poisson problem in § 2.9, it is possible that the collision estimator may perform better from a timing perspective due to the sparsity lost by explicit preconditioning. The deterministic component of the expected value estimator that couples the current state to other local states through the iteration matrix stencil requires application of the estimator to potentially several orders of magnitude more states at every transition event.

For this study, the fuel assembly problem with a 1-group SP_1 discretization was solved using MCSA with the adjoint Monte Carlo solver using both the collision and expected value estimators and both the Richardson and MINRES fixed point iterations. Using the results from the previous section, a Neumann relaxation parameter of 0.7 was used with both estimators and fixed point iterations. For the collision estimator, a Richardson relaxation parameter of 1.1 was used with the Richardson iteration while it was found the expected value estimator had the best performance when a Richardson relaxation parameter of 1.0 was used instead. In addition, ILUT preconditioning with a drop tolerance of 1×10^{-5} and a fill level of 5 was used with a reduced domain approximation fill level of 100 and a threshold of 1×10^{-10} as in the previous analysis. Artificial absorption was also introduced with a value of 0.2 for the MCSA calculations that used the collision estimator as CPU timing performance was improved without a loss in iterative performance while no absorption was used for calculations using the expected value estimator. For each estimator, the number of stochastic histories used to compute the MCSA correction was varied from 25 to 50,000 with the number of MCSA iterations required to converge to a tolerance of 1×10^{-8} recorded for a single eigenvalue iteration along with the CPU time required to achieve convergence.

Figure 3.26 gives the iteration count results and Figure 3.27 gives the timing results. Remarkably, although the fuel assembly problem is significantly more complicated than the simple Poisson problem studied in § 2.9, the results here are largely the same with Figure 3.26 and Figure 2.11 having identical qualitative behavior. Using the expected value estimator with both fixed point iterations creates a relative insensitivity

to the number of histories used to compute the correction. Marginally better iterative and timing performance was observed for the expected value estimator when using the Richardson iteration. Better timing performance is expected as computing the extrapolation parameter in the MINRES iteration requires several additional parallel operations. However, we expect that because the MINRES iteration will converge faster on its own when compared to the Richardson iteration that better iterative performance will be observed when it is accelerated with MCSA. The fact that this was not observed signals that an MCSA scheme leveraging the expected value estimator is also effectively insensitive to the fixed point iteration used. For the collision estimator, the MINRES iteration permits fewer stochastic histories to be used with each MCSA iteration while still maintaining good iterative performance. With respect to CPU time, there is a more pronounced effect due to the larger density of states created by the explicit ILUT preconditioning. Using a reduced domain fill level of 100 creates a situation where the expected value estimator is significantly more expensive per history to compute that the collision estimator due to the coupling of states.

From both Figure 3.26 and Figure 3.27 we see that both estimators and fixed point iterations have comparable iterative and timing performance when 1,000 histories are used at each MCSA iteration. Using 1,000 histories per MCSA iteration, the infinity norm of the residual vector is presented at every MCSA iteration for both estimators and both fixed point iterations in Figure 3.28 as an additional means of comparison. At this number of histories, all MCSA combinations converge the fuel assembly problem monotonically at and approximately the same rate, giving little reason to choose one over the other. The end result is that both estimators provide a viable method for getting good MCSA iterative performance with comparable timing performance. For the collision estimator, using around 20,000 stochastic histories at every iteration gave the best iterative performance while 500 histories per iteration gave the best iterative performance when using the expected value estimator. For purely serial computational performance, there is not a distinguishing feature for the fuel assembly problem presented that would cause one to choose one of the estimators and fixed point iterations over the other as they all have similar minimum CPU times over the range of values tested.

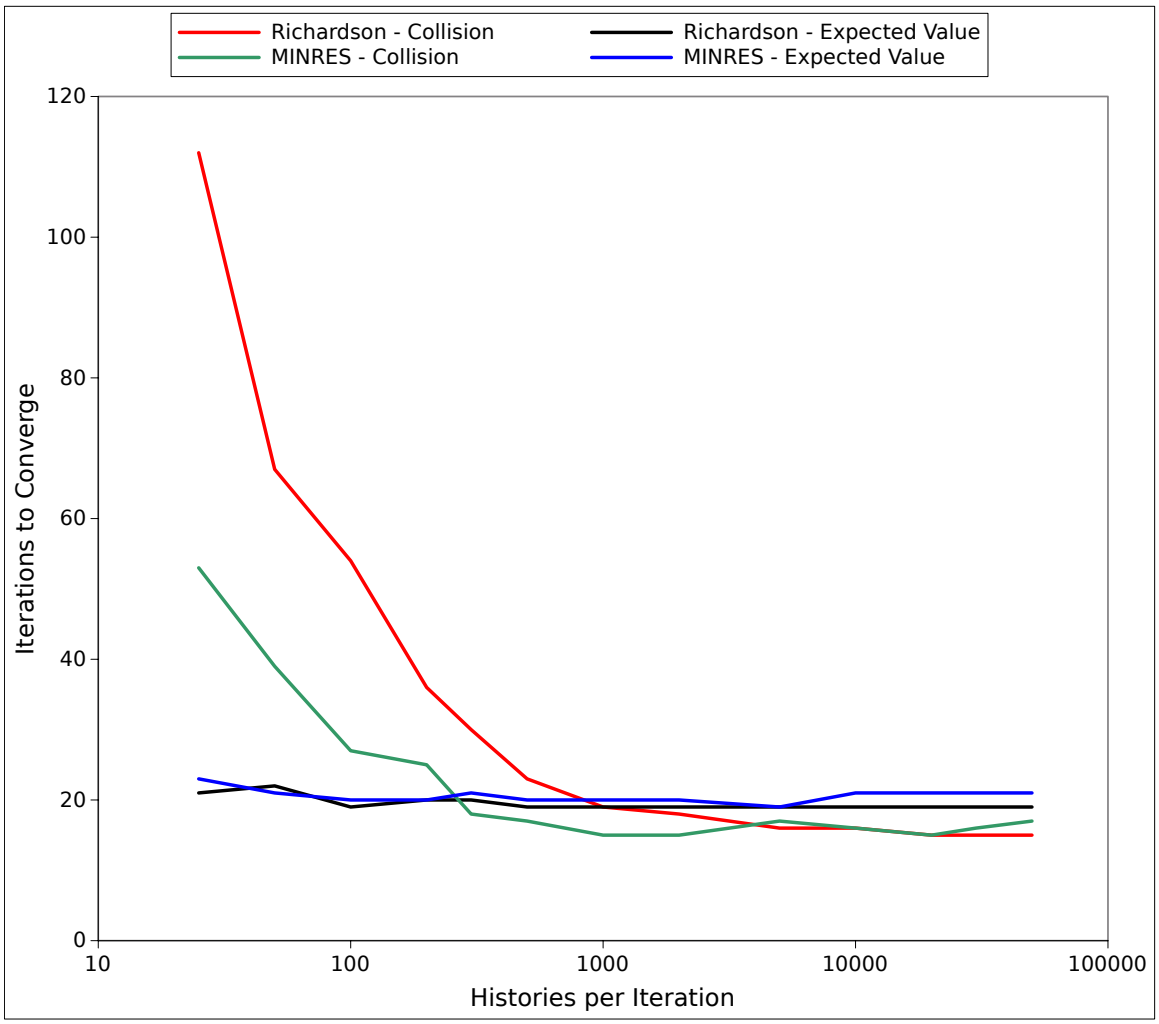

Figure 3.26: **MCSA iterations required to converge the fuel assembly problem to a tolerance of 1×10^{-8}.** *Both the collision and expected value estimators were used with the adjoint Monte Carlo solver to compute the correction at each iteration. Each estimator was used with the Richardson and MINRES fixed point iterations within an MCSA iteration.*

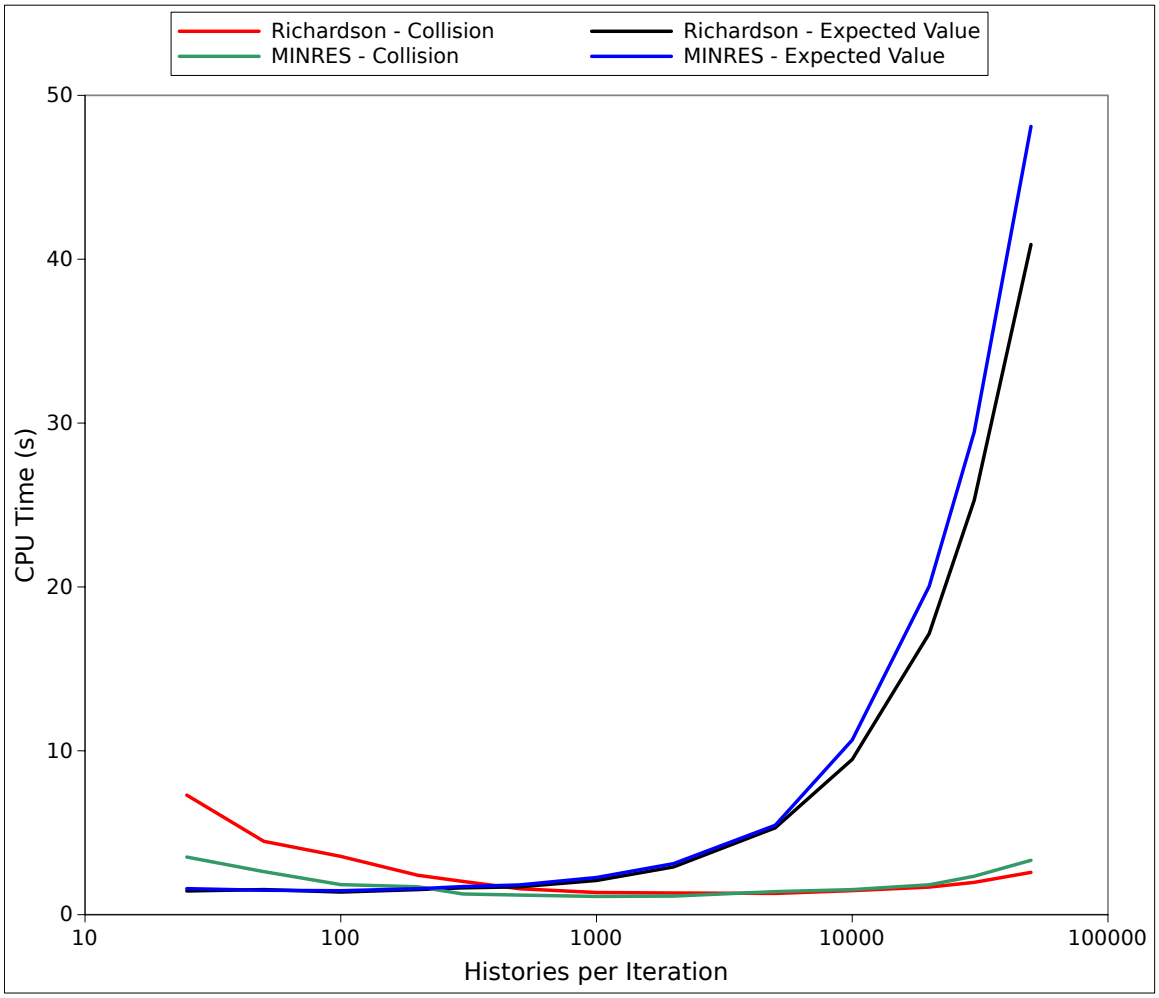

Figure 3.27: **Total CPU time in seconds required to converge the fuel assembly problem to a tolerance of 1×10^{-8}.** *Both the collision and expected value estimators were used with the adjoint Monte Carlo solver to compute the correction at each iteration. Each estimator was used with the Richardson and MINRES fixed point iterations within an MCSA iteration.*

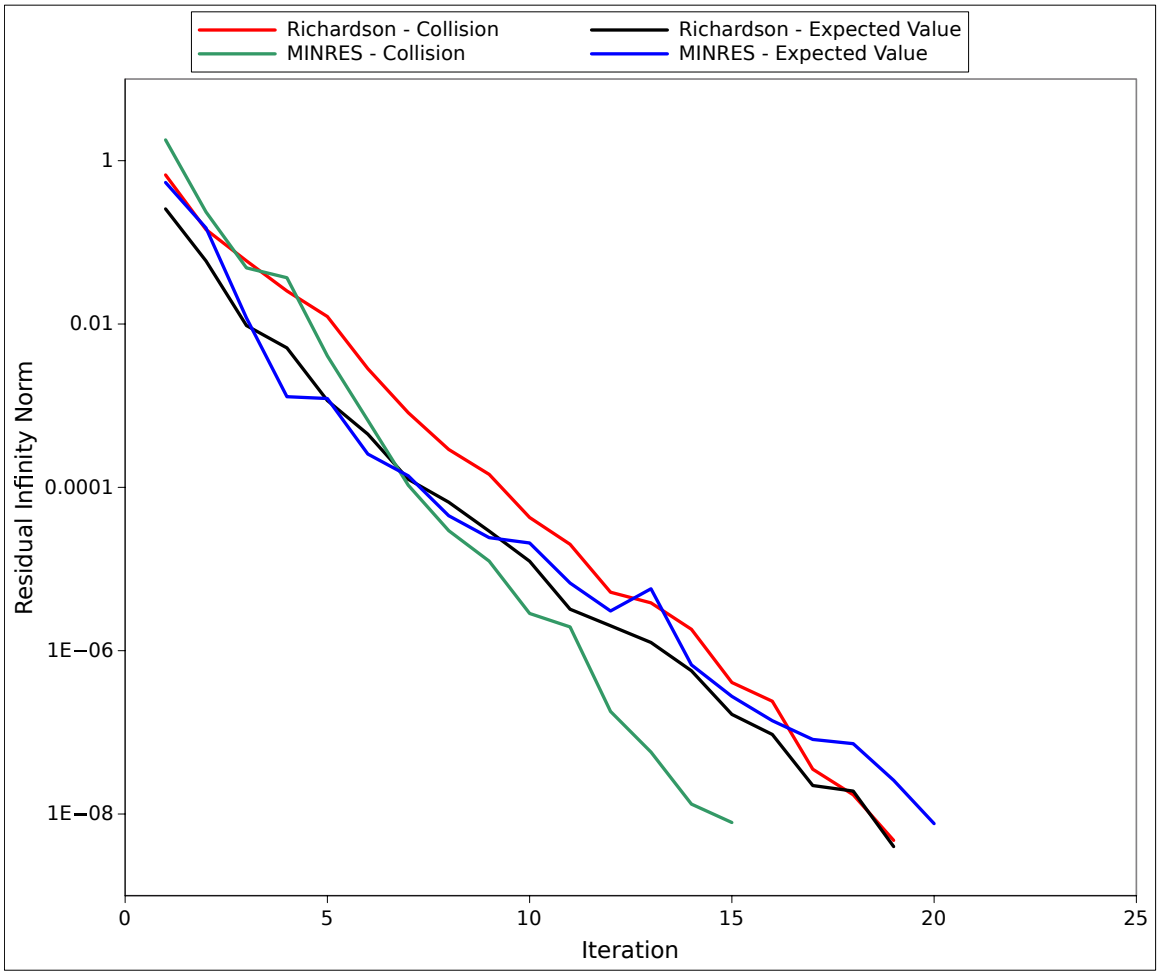

Figure 3.28: **Residual infinity norm as a function of MCSA iteration with 1,000 stochastic histories per iteration.** *Both the collision and expected value estimators were used with the adjoint Monte Carlo solver to compute the correction at each iteration. Each estimator was used with the Richardson and MINRES fixed point iterations within an MCSA iteration.*

3.6 MCSA Verification

Through the numerical studies presented in this chapter, we have developed a set of preconditioning techniques that permit MCSA to be used with the SP_N form of the neutron transport equation and applied to a difficult fuel assembly criticality problem. Along with these preconditioning techniques, a set a varying MCSA parameters were analyzed to find those that yielded the best performance for this particular problem. In addition, several steps were taken to mitigate the dense composite linear operators that arise when performing explicit MCSA preconditioning in order to achieve convergence. Using these results, we compare MCSA directly to conventional linear solvers that would be typically used to solve the SP_N form of the transport equation as presented here. In particular, we will use a pair of production Krylov solvers with which we will compare direct numerical results in order to verify MCSA solutions in this section.

In order to verify the correctness of the MCSA method and implementation used for this work, MCSA solutions to the fuel assembly problem with 1, 2, and 4 energy groups using SP_1 discretization will be directly compared over all group and cell fluxes computed by converging the criticality problem using both BiCGStab and GMRES production implementations from the Trilinos scientific computing library Aztec (Heroux et al., 2005). In addition to comparing the cell and group fluxes, the number of eigenvalue iterations required to converge the problem along with the eigenvalue itself will be compared as a means of verification. If all values computed using MCSA agree with the results from the production Krylov solvers within the solution tolerance, then the MCSA solutions will be deemed correct for these problems.

For the verification calculations, all solvers will be preconditioned with ILUT using a drop tolerance of 1×10^{-5} and a fill level of 5. For GMRES, no restrictions were placed on the size of the subspace and therefore no restarts occurred. When the collision estimator was used with MCSA, 2×10^4 stochastic histories were used to compute the correction for every energy group in the problem (4×10^4 and 8×10^4 histories total for the 2 and 4 group calculations respectively) corresponding to approximately 1 stochastic history per DOF. When the expected value estimator was used, 5×10^2 stochastic histories used for each energy group (1×10^3 and 1.5×10^3 histories total for the 2 and 4 group calculations) giving approximately 1 stochastic history for every 5 DOFs. The reduced domain approximation was also applied in conjunction with the ILUT preconditioning using a fill level of 100 and a threshold value of 1×10^{-10} to reduce the density of states in the Monte Carlo problem. For relaxation parameters, all MCSA computations used a Neumann relaxation parameter of 0.7 while a Richardson

relaxation of 1.1 was used with the collision estimator and 1.0 with the expected value estimator as determined by the previous analysis of the relaxation parameters. Table 3.9 gives definitions for the solvers used to generate the results in the remainder of this section. For the energy group structures, Table 3.10 gives the lower bounds of each group in eV (an implicit upper bound of 2×10^6 eV is assumed for group 0).

Name	**Definition**
BiCGStab-ILUT	BiCGStab preconditioned with ILUT
GMRES-ILUT	GMRES preconditioned with ILUT
MCSA-ILUT-R-C	MCSA preconditioned with ILUT using Richardson fixed point iteration and collision estimator
MCSA-ILUT-MR-C	MCSA preconditioned with ILUT using MINRES fixed point iteration and collision estimator
MCSA-ILUT-R-EV	MCSA preconditioned with ILUT using Richardson fixed point iteration and expected value estimator
MCSA-ILUT-MR-EV	MCSA preconditioned with ILUT using MINRES fixed point iteration and expected value estimator

Table 3.9: **Solver definitions used for MCSA verification and performance analysis.** *The ILUT preconditioner parameters were identical for all calculations and solvers.*

Number of Groups	**Lower Bounds (eV)**
1	$\{ 1 \times 10^{-5} \}$
2	$\{ 1 \times 10^{-1}, 1 \times 10^{-5} \}$
4	$\{ 1 \times 10^{1}, 1 \times 10^{0}, 1 \times 10^{-1}, 1 \times 10^{-5} \}$

Table 3.10: **Energy group lower bounds for the multigroup fuel assembly criticality problem in electron volts.** *An implicit upper bound of* 2×10^6 *eV is assumed for group zero.*

For every combination of energy group structure and solver, the eigenvalue problem was converged with an eigensolver tolerance of 1×10^{-6} and a linear solver tolerance of 1×10^{-8} in a serial calculation. Table 3.11 gives the eigenvalue computed by each solver for each problem as well as the number of eigenvalue iterations required to converge. For every problem, each solver converged to the same eigenvalue within the tolerance of the eigensolver in the same number of eigenvalue iterations.

For each converged eigenvalue calculation, the scalar flux in each energy group was compared over the entire spatial domain with the BiCGStab-ILUT solution used as

Solver	Number of Groups	Iterations	Eigenvalue
BiCGStab-ILUT	1	25	1.1059870
GMRES-ILUT	1	25	1.1059870
MCSA-ILUT-R-C	1	25	1.1059870
MCSA-ILUT-MR-C	1	25	1.1059870
MCSA-ILUT-R-EV	1	25	1.1059870
MCSA-ILUT-MR-EV	1	25	1.1059870
BiCGStab-ILUT	2	35	1.1552390
GMRES-ILUT	2	35	1.1552390
MCSA-ILUT-R-C	2	35	1.1552390
MCSA-ILUT-MR-C	2	35	1.1552390
MCSA-ILUT-R-EV	2	35	1.1552390
MCSA-ILUT-MR-EV	2	35	1.1552390
BiCGStab-ILUT	4	31	1.1758500
GMRES-ILUT	4	31	1.1758500
MCSA-ILUT-R-C	4	31	1.1758500
MCSA-ILUT-MR-C	4	31	1.1758500
MCSA-ILUT-R-EV	4	31	1.1758500
MCSA-ILUT-MR-EV	4	31	1.1758500

Table 3.11: **Serial eigensolver verification results for the fuel assembly problem.** *Each calculation was converged with the same eigensolver and linear solver tolerances. Table 3.9 gives the description for each solver type presented.*

the reference to compute an absolute difference of the flux in each mesh cell. For the 1 group calculations, these absolute differences are plotted spatially along the center-line of the fuel assembly in Figure 3.29. Interestingly, except for the MCSA-ILUT-MR-C calculation, all other 1-group calculations including GMRES had a maximum global flux difference of 9.313×10^{-10} even though different spatial results were observed for all cases. For the MCSA-ILUT-MR-C calculation, a slightly larger maximum error of 2.205×10^{-9} was observed for reasons that remain unexplained. All 1-group calculations have a minimum observed absolute difference of 0 with respect to the BiCGStab calculation.

Table 3.12, Table 3.13, and Table 3.14 give the maximum and minimum absolute differences of the cell scalar fluxes for all calculations and all energy groups using the BiCGStab calculation as the reference. All comparisons were performed within a group such that the differences for the MCSA-ILUT-R-C results in group 3 for the 4-group calculation were computed with the BiCGStab-ILUT solution in group 3 for the 4-group calculation as a reference. For all group flux results given in the table,

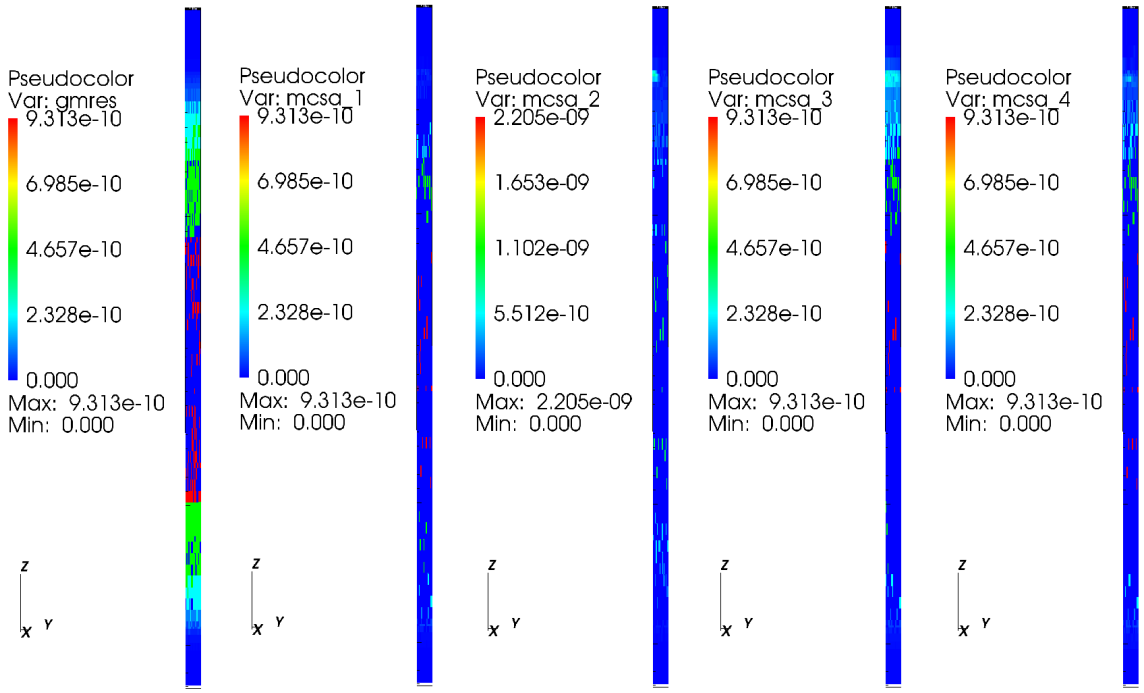

Figure 3.29: **Absolute difference of the group 0 scalar flux between the Aztec BiCGStab calculations and Aztec GMRES and the MCSA variations for the 1-group fuel assembly criticality calculation.** *Left to right: GMRES-ILUT, MCSA-ILUT-R-C, MCSA-ILUT-MR-C, MCSA-ILUT-R-EV, MCSA-ILUT-MR-EV. The fluxes are plotted on a plane intersecting the center-line of the assembly. Table 3.9 gives the description for each solver type presented.*

the maximum differences for all cases was less than the linear solver tolerance used to converge the flux at each eigenvalue iteration meaning that all values over the entire energy and spatial domain are within these limits. In addition, the minimum error for all calculations was observed to be zero, meaning that at least one mesh cell in all calculations for all energy groups the scalar flux values computed by all solvers were identical. Based on the flux results and the eigenvalue results presented here, the MCSA results should be deemed correct for the fuel assembly problem for serial calculations with multiple energy groups.

Solver	Group	Minimum Difference	Maximum Difference
GMRES-ILUT	0	0.0	9.313×10^{-10}
MCSA-ILUT-R-C	0	0.0	9.313×10^{-10}
MCSA-ILUT-MR-C	0	0.0	2.205×10^{-9}
MCSA-ILUT-R-EV	0	0.0	9.313×10^{-10}
MCSA-ILUT-MR-EV	0	0.0	9.313×10^{-10}

Table 3.12: **Serial scalar flux verification results for the 1-group fuel assembly problem.** *The maximum and minimum absolute differences over the entire spatial domain are presented with BiCGStab used as the reference. Each calculation was converged with the same eigensolver and linear solver tolerances. Table 3.9 gives the description for each solver type presented.*

Solver	Group	Minimum Difference	Maximum Difference
GMRES-ILUT	0	0.0	2.794×10^{-9}
MCSA-ILUT-R-C	0	0.0	9.313×10^{-10}
MCSA-ILUT-MR-C	0	0.0	9.313×10^{-10}
MCSA-ILUT-R-EV	0	0.0	9.313×10^{-10}
MCSA-ILUT-MR-EV	0	0.0	9.313×10^{-10}
GMRES-ILUT	1	0.0	4.657×10^{-10}
MCSA-ILUT-R-C	1	0.0	1.364×10^{-10}
MCSA-ILUT-MR-C	1	0.0	1.261×10^{-9}
MCSA-ILUT-R-EV	1	0.0	2.474×10^{-10}
MCSA-ILUT-MR-EV	1	0.0	5.857×10^{-10}

Table 3.13: **Serial scalar flux verification results for the 2-group fuel assembly problem.** *The maximum and minimum absolute differences over the entire spatial domain are presented with BiCGStab used as the reference. Each calculation was converged with the same eigensolver and linear solver tolerances. Table 3.9 gives the description for each solver type presented.*

Solver	Group	Minimum Difference	Maximum Difference
GMRES-ILUT	0	0.0	9.313×10^{-10}
MCSA-ILUT-R-C	0	0.0	1.397×10^{-9}
MCSA-ILUT-MR-C	0	0.0	1.870×10^{-9}
MCSA-ILUT-R-EV	0	0.0	1.397×10^{-9}
MCSA-ILUT-MR-EV	0	0.0	1.869×10^{-9}
GMRES-ILUT	1	0.0	5.821×10^{-11}
MCSA-ILUT-R-C	1	0.0	1.164×10^{-10}
MCSA-ILUT-MR-C	1	0.0	1.164×10^{-10}
MCSA-ILUT-R-EV	1	0.0	1.164×10^{-10}
MCSA-ILUT-MR-EV	1	0.0	1.164×10^{-10}
GMRES-ILUT	2	0.0	1.164×10^{-10}
MCSA-ILUT-R-C	2	0.0	2.328×10^{-10}
MCSA-ILUT-MR-C	2	0.0	2.328×10^{-10}
MCSA-ILUT-R-EV	2	0.0	2.328×10^{-10}
MCSA-ILUT-MR-EV	2	0.0	2.328×10^{-10}
GMRES-ILUT	3	0.0	1.164×10^{-10}
MCSA-ILUT-R-C	3	0.0	1.746×10^{-10}
MCSA-ILUT-MR-C	3	0.0	2.256×10^{-9}
MCSA-ILUT-R-EV	3	0.0	2.801×10^{-10}
MCSA-ILUT-MR-EV	3	0.0	2.256×10^{-10}

Table 3.14: **Serial scalar flux verification results for the 4-group fuel assembly problem.** *The maximum and minimum absolute differences over the entire spatial domain are presented with BiCGStab used as the reference. Each calculation was converged with the same eigensolver and linear solver tolerances. Table 3.9 gives the description for each solver type presented.*

3.7 MCSA Performance Comparison to Conventional Methods

In the previous section, numerical results using MCSA for the fuel assembly calculation were directly compared with production Krylov methods in order to verify their correctness. Deemed correct, performance of the algorithm for serial computations will now be addressed. With the same set of solver parameters used for verification, MCSA will now be directly compared to the same production Krylov solvers in terms of both iterative performance and CPU timing for solutions to the fuel assembly problem. This performance comparison is necessary in order to assess the feasibility of the method and identify areas that require further research and development.

To compare iterative performance, the number of linear solver iterations required to converge each eigenvalue iteration was recorded and then averaged over all eigenvalue iterations for each solver. Figure 3.30 gives the average number of linear solver iterations required to converge a single eigenvalue iteration of the fuel assembly problem as a function of the number of energy groups. In general, the iterative performance of all the solvers tested is comparable with BiCGStab performing the best. We expect this because not only are the multigroup SP_N equations elliptic and positive-definite, but they are also nearly symmetric and therefore we expect the conjugate gradient-based methods to perform well. It also important to note that the iterative performance of each solver is not a strong function of the number of energy groups in the problem (and therefore the problem size).

For MCSA, we are linearly increasing the number of stochastic histories used to compute the correction at each iteration as the number of energy groups are increased and therefore the work performed for the problem is of $O(N)$. It is also important to note the reduced domain approximation parameters, particularly the fill level, were fixed as the number of energy groups was increased. By fixing the fill level, we are effectively fixing the amount of information contained in the composite linear operator available for the Monte Carlo problem. Limiting this information to 100 entries per row in the preconditioned system did not appear to have a significant effect on the iterative performance of the solver. For larger numbers of energy groups (and therefore DOFs) this may be an issue and this number may need to be increased to maintain iterative performance. Currently, memory restrictions arising from the explicit preconditioning strategy prevent larger energy groups to be analyzed using an appropriate preconditioner for the fuel assembly problem.

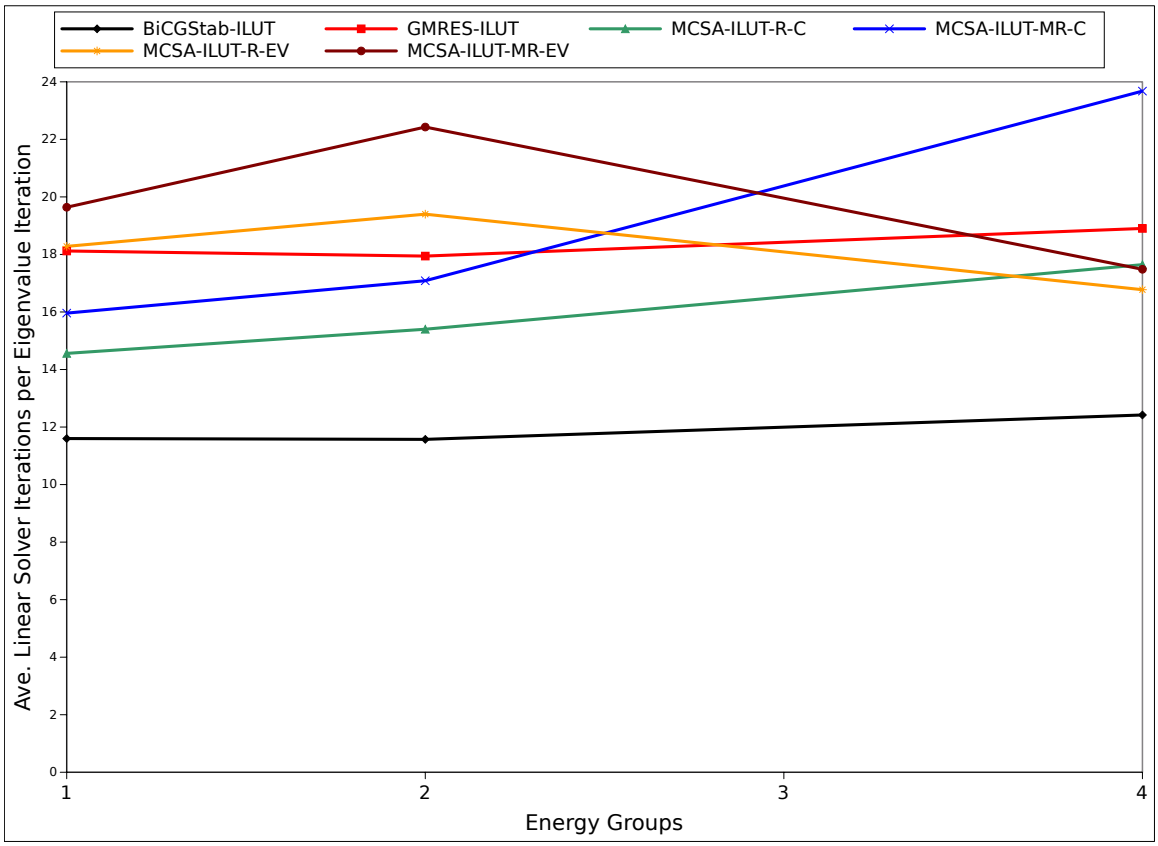

Figure 3.30: **Average number of iterations required to converge each eigenvalue iteration for the fuel assembly problem as a function of energy groups.** *Table 3.9 gives the description for each solver type presented in the legend.*

Although iterative performance for MCSA was comparable to that observed for production Krylov implementations using exactly the same preconditioning scheme, timing performance is not as competitive. For this comparison, the amount of time to perform each linear solver iteration in every eigenvalue iteration was recorded and then averaged over all linear solver iterations. Figure 3.31 gives the average CPU time per linear solver iteration required to converge the fuel assembly problem with all linear solver iterations over all eigenvalue iterations used to compute the average. The production Krylov solver implementations are approximately an order of magnitude faster than the MCSA implementations presented here. We expect these results for several reasons. First, even when the reduced domain approximation is used there are $O(100)$ elements in each row of the system and each of those elements must be processed during a transition in the Monte Carlo random walk sequence. In addition, although the preconditioned fixed point iterations used in the MCSA sequence do not

use the composite linear operator but rather a sequence of matrix vector multiplies to achieve the same preconditioning effect, they do use the explicitly extracted inverse of the preconditioning matrices which themselves are dense, leading to exceedingly slow computation times even in the fixed point iteration. For the production Krylov methods, the ILUT preconditioners are applied to a vector with two triangular solves, one for each triangular factor. Also, the original linear operator has $O(10)$ elements in each row of the system, an order of magnitude less than that used in the reduced domain composite operator. Second, the implementation presented here has not been optimized and it is likely that several components of the algorithm can be implemented in a more desirable time complexity. The production Krylov solvers presented here have enjoyed decades of professional development and optimization. However, it is most important to note here that the CPU timing data presented in Figure 3.31 shows that all methods have the same time complexity; their differences are merely the manifestation of a large time constant due to the effects of the explicit preconditioning scheme. For transport systems where this is not a problem, the literature explicitly shows that MCSA can be competitive with these production Krylov methods using CPU time as a measure (Evans et al., 2012).

Finally, not considered in Figure 3.31 is the time required to actually form the explicit inverse preconditioner matrices and the composite operator through matrix-matrix multiplication. The timing numbers reported were simply to perform the MCSA iteration procedure with the composite operator already formed. Figure 3.32 additionally presents the CPU times for MCSA convergence with the time to form the inverse of the preconditioners and the composite linear operator through matrix-matrix multiplication amortized over all iterations. As is readily observed, including the costs of these operations increases the MCSA computation time by another order of magnitude.

Based on the performance results in this section, MCSA shows promise as a competitive and perhaps even superior method for solutions to the neutron transport problem discretized with the SP_N approximation. Not only are the correct answers produced when compared to production linear solvers, but the iterative performance is comparable to production Krylov methods with identical preconditioning that would typically be used every-day calculations. From a CPU timing perspective, the methods yield the same time complexity as a function of the number of energy groups in the problem when compared to the production Krylov methods. Here, a large time constant is generated due to the explicit preconditioning strategy and the generation of dense linear operators as a result. To improve these results and put general MCSA

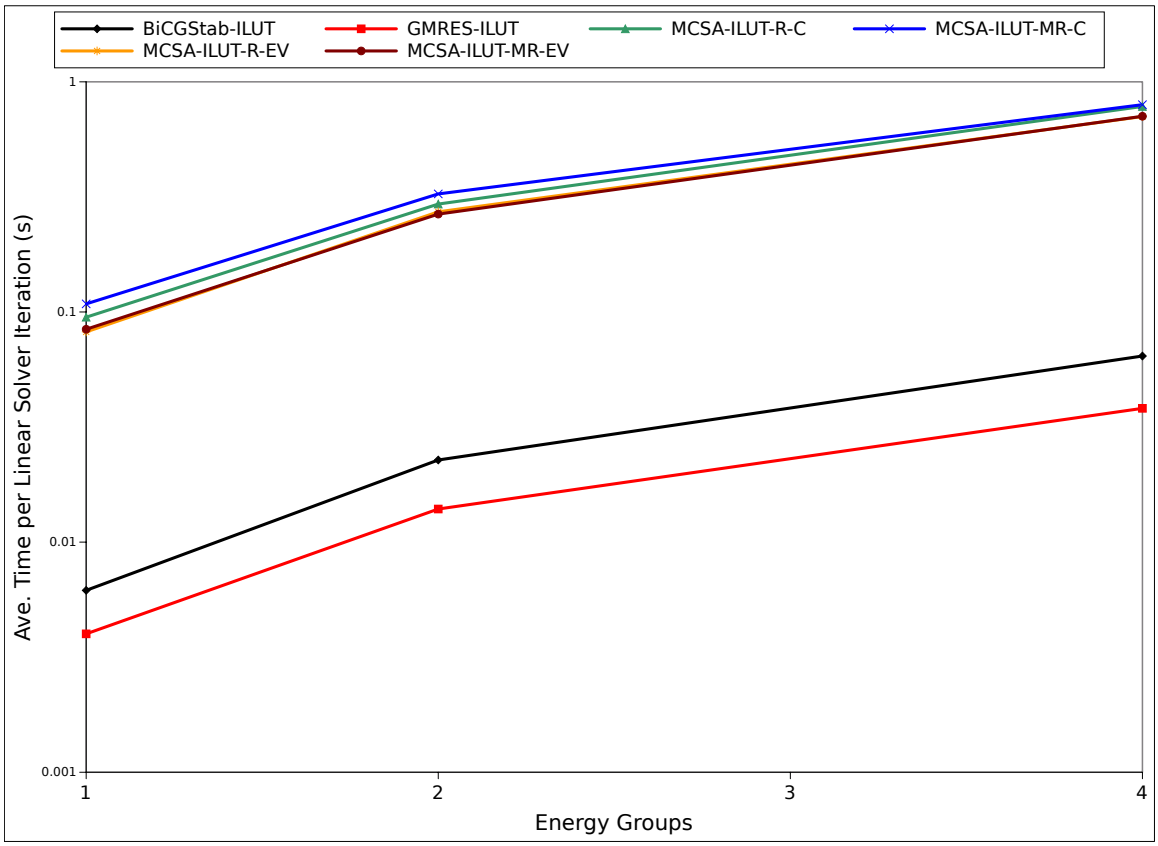

Figure 3.31: **Average CPU time per linear solver iteration in seconds for the fuel assembly problem as a function of energy groups.** *All linear solver iterations over all eigenvalue iterations were used to compute the average. Table 3.9 gives the description for each solver type presented in the legend.*

schemes into a performance regime where they are competitive with Krylov methods for neutron transport problems, significant research will be required to improve upon the explicit preconditioning scheme presented here.

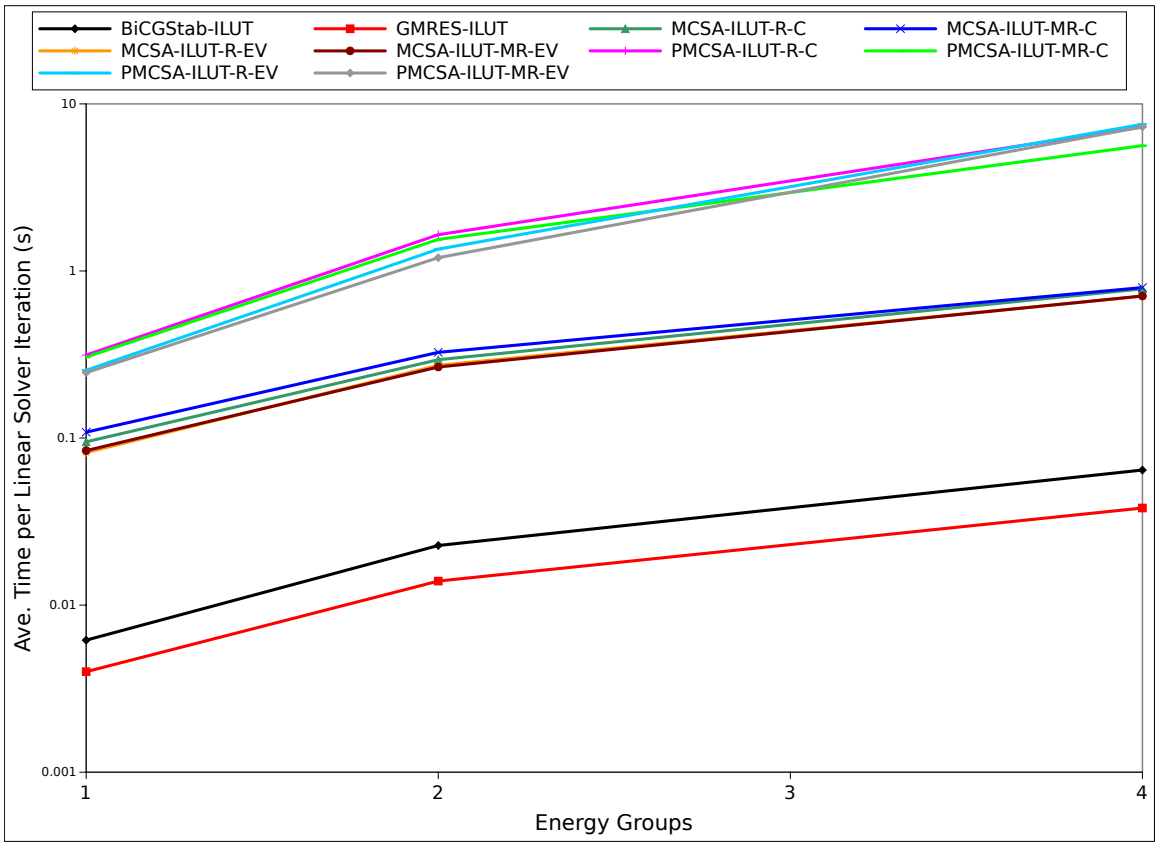

Figure 3.32: **Average CPU time per iteration in seconds for the fuel assembly problem as a function of energy groups with preconditioning time included for the MCSA methods.** *All linear solver iterations over all eigenvalue iterations were used to compute the average. Table 3.9 gives the description for each solver type presented in the legend. Data labeled starting with PMCSA is identical to those labeled with MCSA except that they additionally include the cost of generating the inverse of the preconditioners and composite linear operator.*

Chapter 4

Monte Carlo Synthetic Acceleration Methods for the Navier-Stokes Equations

Nonlinear transport problems are a common occurrence in single and multiple physics problems. Systems of partial differential equations such as those that describe fluid flow or more general transport processes when discretized by conventional methods yield discrete sets of stiff equations with nonlinearities present in the variables. Traditionally, such systems have been solved by linearizing them in a form where the nonlinearities in the variables are eliminated and more traditional linear methods can be used for solutions. Often characterized as segregated methods where physics operators are split and their action on the system approximated in steps, such methods lack consistency and accuracy in resolving the nonlinear component of the solution. In the last 30 years, fully consistent nonlinear methods based on Newton's method have become more popular and many advances have been made in the computational physics field to employ these methods.

In the context of solving standalone linear systems, Monte Carlo methods do not provide significant merit over Krylov methods due to the fact that the linear operator must be explicitly formed. For many applications, such a requirement is prohibitive and perhaps not even feasible to implement. Therefore, a Monte Carlo solver is best suited for situations in which not only are Krylov methods applicable, but also in which the operator is readily, if not naturally, formed. Modern nonlinear methods meet both of these requirements with Newton methods used in conjunction with Krylov methods for a robust solution strategy. Furthermore, modern techniques exist that permit the automatic construction of the linear operator generated within a Newton method based on the nonlinear residual evaluations, providing all of the components necessary for a Monte Carlo solver to provide value. We therefore devise a new nonlinear method based on the MCSA algorithm and Newton's method and discuss its potential benefits.

4.1 Preliminaries

We formulate the *nonlinear problem* as follows (Knoll and Keyes, 2004):

$$\mathbf{F}(\mathbf{u}) = \mathbf{0}\,, \tag{4.1}$$

where $\mathbf{u} \in \mathbb{R}^n$ is the solution vector and $\mathbf{F} : \mathbb{R}^N \to \mathbb{R}^N$ is the function of nonlinear residuals. We write the nonlinear system in this form so that when an exact solution for $\mathbf{u}$ is achieved, all residuals evaluate to zero. *Newton's method* is a root finding algorithm and therefore we can use it to solve Eq (4.1) if we interpret the exact solution $\mathbf{u}$ to be the roots of $\mathbf{F}(\mathbf{u})$. Newton's method is also an iterative scheme, and we can generate this procedure by building the Taylor expansion of the $k+1$ iterate of the nonlinear residuals about the previous k iterate:

$$\mathbf{F}(\mathbf{u}^{k+1}) = \mathbf{F}(\mathbf{u}^k) + \mathbf{F}'(\mathbf{u}^k)(\mathbf{u}^{k+1} - \mathbf{u}^k) + \frac{\mathbf{F}''(\mathbf{u}^k)}{2}(\mathbf{u}^{k+1} - \mathbf{u}^k)^2 + \cdots\,. \tag{4.2}$$

If we ignore the nonlinear terms in the expansion and assert that at the $k+1$ iterate $\mathbf{u}^{k+1}$ is the exact solution such that $\mathbf{F}(\mathbf{u}^{k+1}) = \mathbf{0}$, then we are left with the following equality:

$$-\mathbf{F}(\mathbf{u}^k) = \mathbf{F}'(\mathbf{u}^k)(\mathbf{u}^{k+1} - \mathbf{u}^k)\,. \tag{4.3}$$

We note two things of importance in Eq (4.3). The first is that $\mathbf{F}'(\mathbf{u}^k)$ is in fact the *Jacobian*, $\mathbf{J}(\mathbf{u})$, of the set of nonlinear residuals and is defined element-wise as:

$$J_{ij} = \frac{\partial F_i(\mathbf{u})}{\partial u_j}\,. \tag{4.4}$$

Second, we note that $(\mathbf{u}^{k+1} - \mathbf{u}^k)$ is simply the solution update from the k iterate to the $k+1$ iterate. We will define this update as the *Newton correction* at the k iterate, $\delta\mathbf{u}^k$. To finish, we can then rearrange Eq (4.3) to define the Newton iteration scheme for nonlinear problems:

$$\mathbf{J}(\mathbf{u})\delta\mathbf{u}^k = -\mathbf{F}(\mathbf{u}^k) \tag{4.5a}$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \delta\mathbf{u}^k\,. \tag{4.5b}$$

There are then three distinct steps to perform: evaluation of the nonlinear residuals using the solution at the k iterate, the solution of a linear system to compute the Newton correction where the Jacobian matrix of the nonlinear equation set is the

linear operator, and the application of the correction to the previous iterate's solution to arrive at the next iterate's solution. In the asymptotic limit, the iterations of Newton's method will converge the nonlinear residual quadratically (Kelley, 1995). Convergence criteria is set for stopping the iteration sequence based on the nonlinear residual. Commonly, the following criteria is used:

$$||\mathbf{F}(\mathbf{u}^k)|| < \epsilon||\mathbf{F}(\mathbf{u}^0)|| \,, \tag{4.6}$$

where ϵ is a user defined tolerance parameter. Newton's method is *consistent* in that all components of the nonlinear functions that describe the physics we are modeling are updated simultaneously in the iteration sequence with respect to one another. This is in comparison to *inconsistent* strategies, such as a pressure correction strategy for solving the Navier-Stokes equations (Pletcher et al., 1997), where the components of $\mathbf{u}$ are updated in a staggered fashion depending on the particular equations that they are associated with.

4.2 The FANM Method

In production physics codes based on nonlinear equations sets, Newton-Krylov methods are the primary means of generating a fully consistent solution scheme (Evans et al., 2006, 2007; Gaston et al., 2009; Godoy and Liu, 2012). Typically, for large scale simulations these problems are memory limited due to the subspaces generated by robust Krylov methods. Often, a matrix-free approach is chosen to relax memory requirements over directly generating the Jacobian matrix and facilitate the implementation. However, as we observed in previous sections, these matrix-free methods suffer from poorly scaled problems and the first order error introduced by the Jacobian approximation. In addition, it was observed that the savings induced by the matrix-free approach is eventually amortized over a number of nonlinear iterations where it becomes more efficient computationally to instead form the Jacobian.

In Chapter 2, we focused our efforts on developing and improving Monte Carlo methods for inverting linear systems. These methods, when used to accelerate a stationary method in MCSA, enjoy a robust implementation and exponential convergence rates. Further, the only storage required is that of the full linear system including the linear operator so that we may generate transition probabilities for the random walk sequence. Although this requires more storage to represent the linear system than that of a Krylov method where the operator is not required, we

do not incur any additional storage costs once the iteration sequence begins. In the context of nonlinear problems, the Jacobian matrix that we are required to generate for the Monte Carlo solvers may be generated at will from the nonlinear functions in the Newton system using automatic differentiation. Not only do we then have a simple and automated way to generate the Jacobian, but we also enjoy a Jacobian of numerical precision equivalent to that of our function evaluations. We therefore propose the *Forward-Automated Newton-MCSA* (FANM) method that utilizes all of the above components. We hypothesize that such a method will be competitive with Newton-Krylov methods not only from a convergence and timing perspective, but also relax scaling requirements of matrix-free methods and memory costs of both matrix-free and fully formed Jacobian methods to allow the application developer to solve problems of finer discretization and higher-order functional representations while maintaining a robust and efficient parallel implementation.

4.2.1 Parallel FANM Implementation

A parallel FANM method relies on a basic set of parallel matrix-vector operations as well as the global residual and Jacobian assembly procedure described in § **??**. Consider the Newton iteration scheme in Eq (4.5). We must first assemble the linear system in parallel through the element-wise function evaluations to generate both the global Jacobian operator and the global residual vector on the right hand side. Per Bartlett's work, efficient and automated parallel mechanisms are available to do this through a sequence of scatter/gather operations. With these tools available for residual and Jacobian generation, the remainder of the parallel procedure is simple, with the linear Newton correction system solved using the parallel MCSA method as previously described and the Newton correction applied to the previous iterate's solution through a parallel vector update operation.

As Newton methods are formulated independent of the inner linear solver generating the Newton corrections, the actual performance of the nonlinear iterations using MCSA is expected to be similar to that of traditional Newton-Krylov methods. Furthermore, we expect to achieve numerically identical answers with a Newton-MCSA method as other Newton methods and we should indeed verify this. The parallel performance of such a method will inherently be bound to the parallel Monte Carlo implementation of the linear solver as the parallel operations at the nonlinear iteration level are identical to those that you would perform with a Newton-Krylov method. Matrix-free formulations will have different parallel performance than these methods, and therefore

we should compare FANM performance to JFNK-based schemes. More important than performance in this situation is the reduced memory pressure that a FANM implementation provides, as discussed in § **??**, because a FANM method will not generate a subspace in the linear solver and compressed storage for sparse matrices are utilized, we expect significant memory savings over Newton-Krylov methods. We must measure the memory utilization of both of these methods in order to quantify their differences and provide additional analysis of the FANM method's merits, or lack thereof.

4.3 Navier-Stokes Benchmark Problems

To verify the FANM method for nonlinear problems, we choose benchmark solutions for the 2-dimensional, steady, incompressible Navier-Stokes equations on a rectilinear grid in much the same way as Shadid and Pawlowski's work on Newton-Krylov methods for the solution of these equations (Shadid et al., 1997; Pawlowski et al., 2006). We define these equations as follows:

$$\rho\mathbf{u}\cdot\nabla\mathbf{u} - \nabla\cdot\mathbf{T} - \rho\mathbf{g} = \mathbf{0} \tag{4.7a}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{4.7b}$$

$$\rho C_p\mathbf{u}\cdot\nabla T + \nabla\cdot\mathbf{q} = 0\,, \tag{4.7c}$$

where ρ is the fluid density, $\mathbf{u}$ is the fluid velocity, $\mathbf{g}$ gravity, C_p the specific heat capacity at constant pressure of the fluid and T the temperature of the fluid. Eq (4.7a) provides momentum transport, Eq (4.7b) provides the mass balance, and Eq (4.7c) provides energy transport with viscous dissipation effects neglected. In addition, we close the system with the following equations:

$$\mathbf{T} = -P\mathbf{I} + \mu[\nabla\mathbf{u} + \nabla\mathbf{u}^T] \tag{4.8a}$$

$$\mathbf{q} = -k\nabla T\,, \tag{4.8b}$$

where $\mathbf{T}$ is the stress tensor, P is the hydrodynamic pressure, μ is the dynamic viscosity of the fluid, $\mathbf{q}$ is the heat flux in the fluid, and k is the thermal conductivity of the fluid. This set of strongly coupled equations possesses both the nonlinearities and asymmetries that we are seeking for qualification of the FANM method. Further, physical parameters within these equations can be tuned to enhance the nonlinearities. We will then apply these equations to the following three standard benchmark problems.

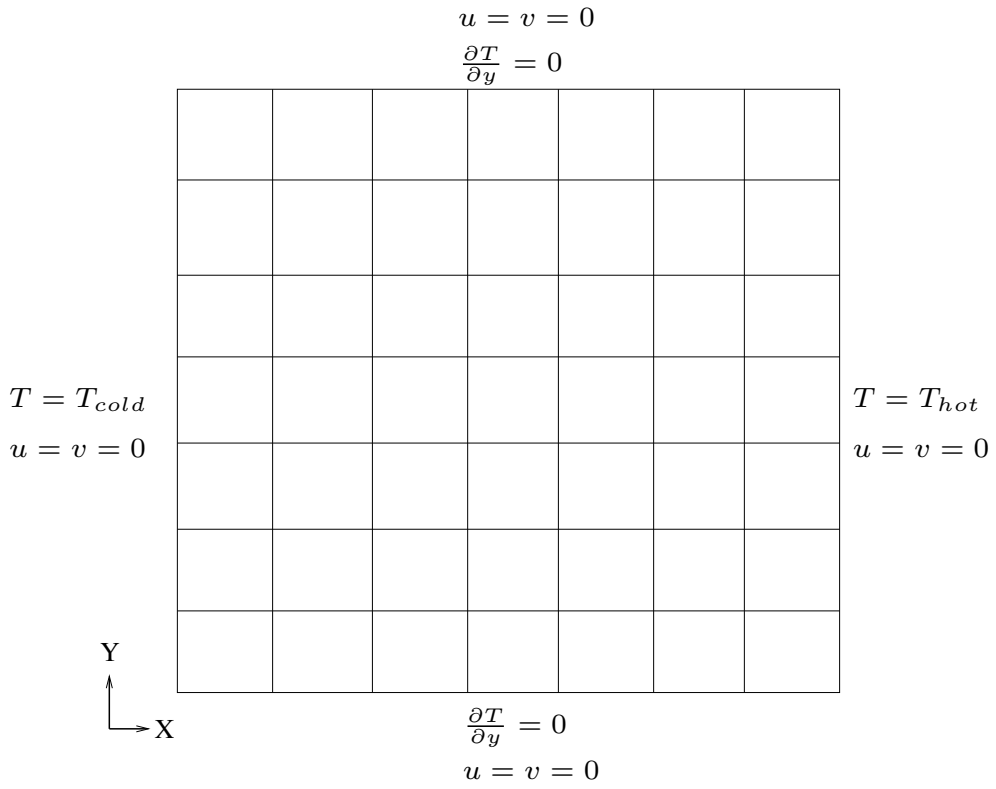

Figure 4.1: **Problem setup for the natural convection cavity benchmark.** *Dirichlet conditions are set for the temperature on the left and right while Neumann conditions are set on the top and bottom of the Cartesian grid. The temperature gradients will cause buoyancy-driven flow. Zero velocity Dirichlet conditions are set on each boundary. No thermal source was present.*

4.3.1 Thermal Convection Cavity Problem

In 1983 a benchmark solution for the natural convection of air in a square cavity was published (De Vahl Davis, 1983) as shown in Figure 4.1 for the solution of the energy, mass, and momentum equations. In this problem, a rectilinear grid is applied to the unit square. No heat flow is allowed out of the top and bottom of the square with a zero Neumann condition specified. Buoyancy driven flow is generated by the temperature gradient from the cold and hot Dirichlet conditions on the left and right boundaries of the box. By adjusting the Rayleigh number of the fluid (and therefore adjusting the ratio of convective to conductive heat transfer), we can adjust the influence of the nonlinear convection term in Eq (4.7a). In Shadid's work, Rayleigh numbers of up to 1×10^6 were used for this benchmark on a 100×100 square mesh grid.

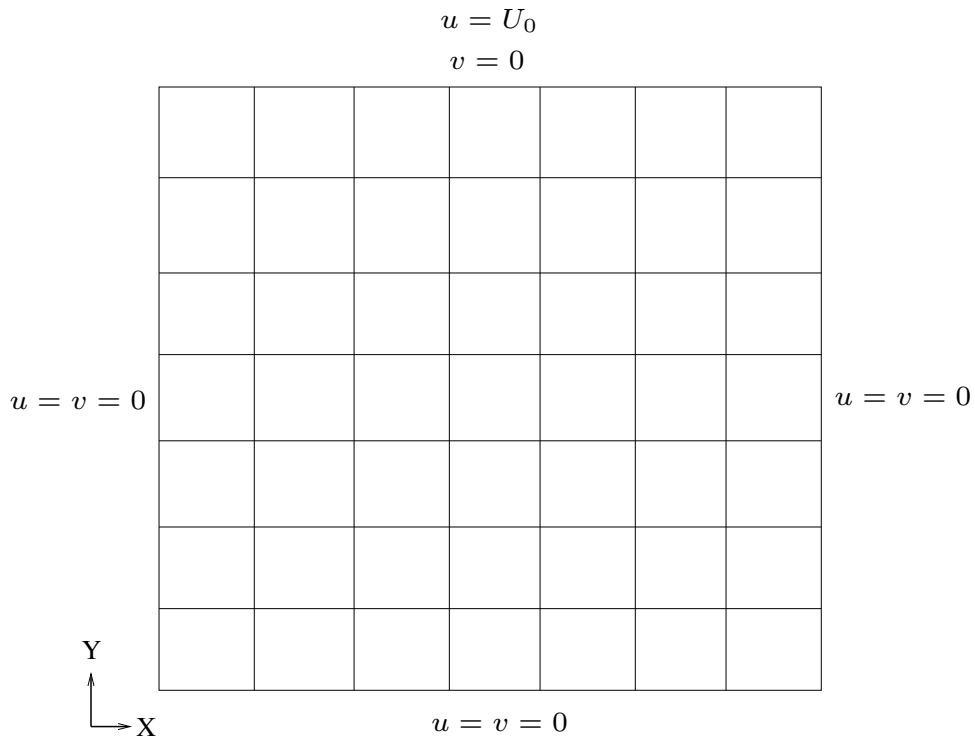

Figure 4.2: **Problem setup for the lid driven cavity benchmark.** *Dirichlet conditions of zero are set for the velocity on the left and right and bottom while the Dirichlet condition set on the top provides a driving force on the fluid.*

4.3.2 Lid Driven Cavity Problem

As an extension of the convection problem, the second benchmark problem given by Ghia (Ghia et al., 1982) adds a driver for the flow to introduce higher Reynolds numbers into the system, providing more inertial force to overcome the viscous forces in the fluid. The setup for this problem is equally simple, containing only the Dirichlet conditions as given in Figure 4.2 and is only applied to the mass and momentum equations on the unit square. The top boundary condition will provide a driver for the flow and its variation will in turn vary the Reynolds number of the fluid. An increased velocity will generate more inertial forces in the fluid, which will overcome the viscous forces and again increase the influence of the nonlinear terms in Eq (4.7a). Shadid also used a 100×100 grid for this benchmark problem with Reynolds numbers up to 1×10^4.

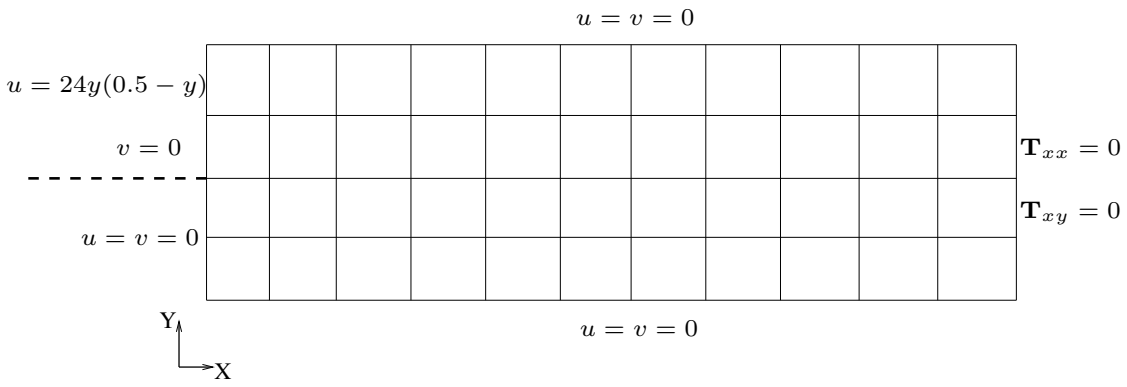

Figure 4.3: **Problem setup for the backward facing step benchmark.** *Zero velocity boundary conditions are applied at the top and bottom of the domain while the outflow boundary condition on the right boundary is represented by zero stress tensor components in the direction of the flow. For the inlet conditions, the left boundary is split such that the top half has a fully formed parabolic flow profile and the bottom half has a zero velocity condition, simulating flow over a step.*

4.3.3 Backward-Facing Step Problem

The third benchmark was generated by Gartling in 1990 and consists of both flow over a backward step and an outflow boundary condition (Gartling, 1990). Using the mass and momentum equations while neglecting the energy equation, this problem utilizes a longer domain with a 1/30 aspect ratio with the boundary conditions as shown in Figure 4.3. In this problem, the inflow condition is specified by a fully-formed parabolic flow profile over a zero velocity boundary representing a step. The flow over this step will generate a recirculating backflow under the inlet flow towards the step. As in the lid driven cavity problem, the nonlinear behavior of this benchmark and the difficulty in obtaining a solution is dictated by the Reynolds number of the fluid. In Shadid's work, a 20×400 non-square rectilinear grid was used to discretize the domain with Reynolds number up to 5×10^2.

4.4 FANM Verification

4.5 FANM Performance Comparison to Conventional Methods

Chapter 5

Parallel Monte Carlo Synthetic Acceleration Methods

For MCSA to be viable at the production scale for nuclear engineering applications, scalable parallel implementations of the algorithm are required. Reviewing the literature, MCSA has yet to be parallelized and the Neumann-Ulam Monte Carlo method on which it is built has only been parallelized through history-level parallelism with full domain replication (Alexandrov, 1998). In order to solve large linear transport systems with MCSA, a domain-decomposed parallel strategy is required. In the formulation of a parallel MCSA algorithm, we recognize that the algorithm occurs in two stages, an outer iteration performing fixed point iteration and applying the correction, and an inner Monte Carlo solver that is generating the correction via the adjoint or forward methods. The parallel aspects of both these components must be considered. Therefore, we will develop and implement parallel algorithms for the Neumann-Ulam and MCSA methods leveraging both the knowledge gained from the general parallel implementations of Krylov methods reviewed in Appendix A and modern parallel strategies for domain decomposed Monte Carlo as developed by the reactor physics community.

In this chapter we briefly review particle transport methods for domain decomposed Monte Carlo. Considering a domain decomposed Neumann-Ulam method, we derive an analytic framework based on algebraic quantities from which to estimate aspects of its performance when applied to neutron transport systems. We then devise a parallel algorithm for the Neumann-Ulam method based on the multiple-set overlapping-domain decomposition algorithm and a parallel algorithm for the MCSA iteration that leverages the parallel Monte Carlo algorithm and general parallel matrix-vector operations. Using parallel solutions for the neutron diffusion problem, we verify the correctness of the parallel algorithm and implementation by comparing the numerical results against production Krylov methods. With the verified implementation of the algorithm, we perform parallel scaling studies to test its performance on a leadership class machine and compare this performance to the production Krylov methods.

5.1 Domain Decomposed Monte Carlo

Large-scale problems will for reasons typically related to memory restrictions or performance have their data partitioned such that each parallel process owns a subset of the equations in the linear system. Given this convention, the Neumann-Ulam Monte Carlo algorithm must perform random walks over a domain that is decomposed and must remain decomposed to avoid the same performance and memory restrictions that required the domain to be decomposed. To parallelize this algorithm, we then seek parallel Monte Carlo algorithms that handle domain decomposition.

To motivate this problem, consider the square domain presented in Figure 5.1 and on this domain we imagine a Monte Carlo particle transport problem. If the domain were decomposed into 9 subdomains as shown, each of those subdomains and their associated data (i.e. cross sections) could be owned by a different parallel process in the computation. In Figure 5.1, the tracks of 3 particles that are born in the center subdomain are shown. In this example, particle A is first transported from the center domain to the domain directly to the left. Before the scattering event in the new domain may be processed, particle A must be communicated between between the two parallel processes that own those domains. For any given particle in the transport simulation, this communication event may hardly occur during the lifetime of the particle (particle B), or may occur many times (particle C) depending on the problem parameters and the random path in phase space taken by the particle. Scalable parallel algorithms for domain decomposed Monte Carlo are those that handle this domain-to-domain communication of particles effectively and balance it with the on-process computations for particle interactions, tallies, response calculations, and other simulation requirements.

For a domain decomposed Neumann-Ulam method, the situation is very much the same where instead the transport process is now discrete and the "physics" of the transport process is the Monte Carlo game with probabilities and weights described by Eq (2.13) for the given linear problem. Figure 5.2 gives the domain decomposed Neumann-Ulam analog of particle transport problem in Figure 5.1. Imagine in this problem that we are solving for the monoenergetic scalar neutron flux as in the model diffusion problem presented in Appendix F. In this problem, the solution has only a spatial dependence and therefore each point in the mesh corresponds to an equation in the resulting linear system. In Figure 5.2, the domain has now been discretized by this mesh and again decomposed into 9 subdomains, each owning a piece of the mesh and therefore the corresponding equations in the linear system. As we play the Monte

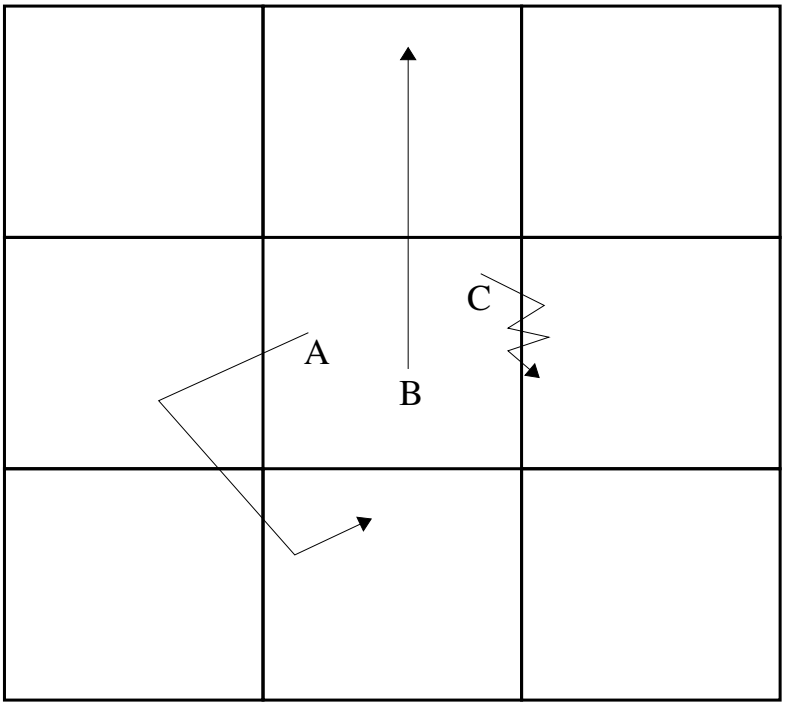

Figure 5.1: **Domain decomposed Monte Carlo transport example illustrating how domain decomposition requires parallel communication of particles.** *If a particle crosses a domain boundary it must be communicated to the next domain in its path.*

Carlo game presented in Chapter 2, the discretization and resulting Neumann-Ulam decomposition of the problem describes how each state in the system is coupled to the other states in the system through the set of equations that form the linear system. This coupling is then responsible for the discrete random walk that each history takes as shown in this example. As discrete states (or rows in the linear system) that do not exist in the local domain are reached during the random walk by stochastic histories, the histories must be communicated to the subdomain that does own the discrete state in an analogous fashion to the particles in the previous example.

To further motivate using a parallel algorithm similar to particle transport methods, again consider history A in Figure 5.2. This history begins in the center subdomain and through the course of transitioning through states in the system arrives at a state that exists in the subdomain directly to the left of the center subdomain. History A must then be communicated from the center subdomain to the subdomain immediately to the left to continue the random walk. As with the particle transport system, we may find that histories in a domain decomposed Neumann-Ulam method are only

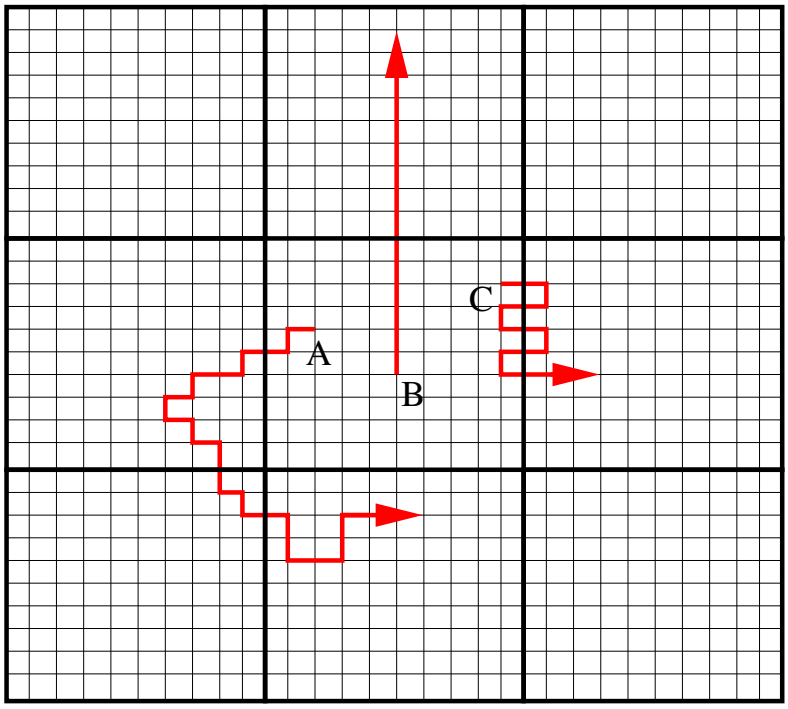

Figure 5.2: **Domain decomposed Neumann-Ulam example illustrating how domain decomposition requires parallel communication of histories.** *Each mesh point corresponds to an equation in the linear system and the coupling among equations is described by the discretization of the problem.*

communicated a few times (history B) if at all, or they may be communicated many times (history C) depending on the discretization and the outcome of the Monte Carlo game. In identical fashion to the particle transport problem, a domain decomposed Neumann-Ulam method has the same communication requirements with all histories in both examples requiring the same parallel operations. The amount of communication of histories from domain to domain will be the primary factor in the parallel scalability of the algorithm. In the next section, the number of histories communicated in a given problem will be quantified analytically for a simple model problem.

5.2 Analytic Framework for Domain-Decomposed Monte Carlo

To date, parallel Neumann-Ulam methods have been limited to full domain replication with parallelism exploited through individual histories (Alexandrov, 1998) and in this chapter we will exploit particle transport algorithms to alleviate this. To accomplish this, we recognize from the literature that stochastic histories must be transported from domain to domain as the simulation progresses and they transition to states that are not in the local domain. Because we have chosen a domain decomposition strategy in a parallel environment, this means that communication of these histories must occur between compute nodes owning neighboring pieces of the global domain. We wish to characterize this communication not only because communication is in general expensive, but also because these nearest-neighbor communication sequences, specifically, have poor algorithmic strong scaling (Gropp et al., 2001) in much the same way as a parallel matrix-vector multiply operation. Therefore, we desire a framework to provide a simple, analytic theory based on the properties of the transport system that will allow for estimates of the domain decomposed behavior of the Neumann-Ulam method in terms of the amount of information that must be communicated.

When solving problems where the linear operator is symmetric, a host of analytic techniques exist based on the eigenvalue spectrum of the operator that characterize their behavior in the context of deterministic linear solvers. Using past work, these techniques are adapted to the domain decomposed Neumann-Ulam method using the one-speed, two-dimensional neutron diffusion equation and spatial discretization presented in Appendix F as a model transport problem. Using the linear system generated by the discretization of the model problem, we use a spectral analysis to generate analytic relations for the eigenvalues of the operator based on system parameters. Using the eigenvalue spectra, we then build relationships to characterize the transport of stochastic histories in a decomposed domain and the fraction of histories that leak from a domain and will therefore have to be communicated. Finally, we compare these analytic results to numerical experiments conducted with the model transport problem.

5.2.1 Spectral Analysis

The convergence of the Neumann series in Eq (2.38) approximated by the Monte Carlo solver is dependent on the eigenvalues of the iteration matrix. We will compute these eigenvalues by assuming eigenfunctions of the form (LeVeque, 2007):

$$\Phi_{p,q}(x,y) = e^{2\pi\imath px} e^{2\pi\imath qy}\,, \tag{5.1}$$

where different combinations of p and q represent the different eigenmodes of the solution. As these are valid forms of the solution, then the action of the linear operator on these eigenfunctions will yield the eigenvalues of the matrix as they exist on the unit circle in the complex plane.

For the model problem, we first compute the eigenvalues for the diffusion operator $\mathbf{D}$ by applying the operator to the eigenfunctions and noting that $x = ih$ and $y = jh$:

$$\begin{aligned}\mathbf{D}\Phi_{p,q}(x,y) = \lambda_{p,q}(\mathbf{D}) = \\ -\frac{D}{6h^2}\Big[&4e^{-2\pi\imath ph} + 4e^{2\pi\imath ph} + 4e^{-2\pi\imath qh} + 4e^{2\pi\imath qh} + e^{-2\pi\imath ph}e^{-2\pi\imath qh} \\ &+ e^{-2\pi\imath ph}e^{2\pi\imath qh} + e^{2\pi\imath ph}e^{-2\pi\imath qh} + e^{2\pi\imath ph}e^{2\pi\imath qh} - 20\Big] + \Sigma_a\,.\end{aligned} \tag{5.2}$$

Using Euler's formula, we can collapse the exponentials to trigonometric functions:

$$\lambda_{p,q}(\mathbf{D}) = -\frac{D}{6h^2}[8\cos(\pi ph) + 8\cos(\pi qh) + 4\cos(\pi ph)\cos(\pi qh) - 20] + \Sigma_a\,. \tag{5.3}$$

As Eq (F.1) is diagonally dominant, point Jacobi preconditioning as outlined in § 2.6.2 is sufficient to reduce the spectral radius of the iteration matrix below unity and therefore ensure convergence of the Neumann series. Applying this preconditioner, we are then solving the following diffusion system:

$$\mathbf{M}^{-1}\mathbf{D}\boldsymbol{\phi} = \mathbf{M}^{-1}\mathbf{s}\,. \tag{5.4}$$

The operator $\mathbf{M}^{-1}\mathbf{D}$ is merely the original diffusion operator with each row scaled by the diagonal component. As we have defined a homogeneous domain, the scaling factor, α, is the same for all rows in the operator and defined as the $\phi_{i,j}$ coefficient from Eq (F.5):

$$\alpha = \left[\frac{10D}{3h^2} + \Sigma_a\right]^{-1}\,. \tag{5.5}$$

Using this coefficient, we then have the following spectrum of preconditioned eigenvalues:

$$\lambda_{p,q}(\mathbf{M}^{-1}\mathbf{D}) = \alpha\lambda_{p,q}(\mathbf{D})\,. \tag{5.6}$$

The spectral radius of the iteration matrix is obtained by seeking its largest eigenvalue. As with the diffusion operator, we can use the same analysis techniques to find the eigenvalues for the iteration matrix. We use a few simplifications by noting that if the Jacobi preconditioned iteration matrix is $\mathbf{H} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{D}$, the we except all terms on the diagonal of the iteration matrix to be zero such that we have the following stencil:

$$\mathbf{H}\boldsymbol{\phi} = \frac{\alpha D}{6h^2}\Big[4\phi_{i-1,j} + 4\phi_{i+1,j} + 4\phi_{i,j-1} + 4\phi_{i,j+1} + \\ \phi_{i-1,j-1} + \phi_{i-1,j+1} + \phi_{i+1,j-1} + \phi_{i+1,j+1}\Big]\,. \tag{5.7}$$

Inserting the eigenfunctions defined by Eq (5.1) we get:

$$\lambda_{p,q}(\mathbf{H}) = \frac{\alpha D}{6h^2}\Big[4e^{-2\pi\imath ph} + 4e^{2\pi\imath ph} + 4e^{-2\pi\imath qh} + 4e^{2\pi\imath qh} + e^{-2\pi\imath ph}e^{-2\pi\imath qh} \\ + e^{-2\pi\imath ph}e^{2\pi\imath qq} + e^{2\pi\imath ph}e^{-2\pi\imath qh} + e^{2\pi\imath ph}e^{2\pi\imath qh}\Big]\,, \tag{5.8}$$

which simplifies to:

$$\lambda_{p,q}(\mathbf{H}) = \frac{\alpha D}{6h^2}[8\cos(\pi ph) + 8\cos(\pi qh) + 4\cos(\pi ph)\cos(\pi qh)]\,, \tag{5.9}$$

giving the eigenvalue spectrum for the Jacobi preconditioned iteration matrix. To find the maximum eigenvalue, Eq (5.6) is plotted as a function of p with $p = q$ in Figure 5.3 for various values of Σ_a. We find that the maximum eigenvalue exists when $p = q = 0$, giving the following for the spectral radius of the Jacobi preconditioned iteration matrix:

$$\rho(\mathbf{H}) = \frac{10\alpha D}{3h^2}\,. \tag{5.10}$$

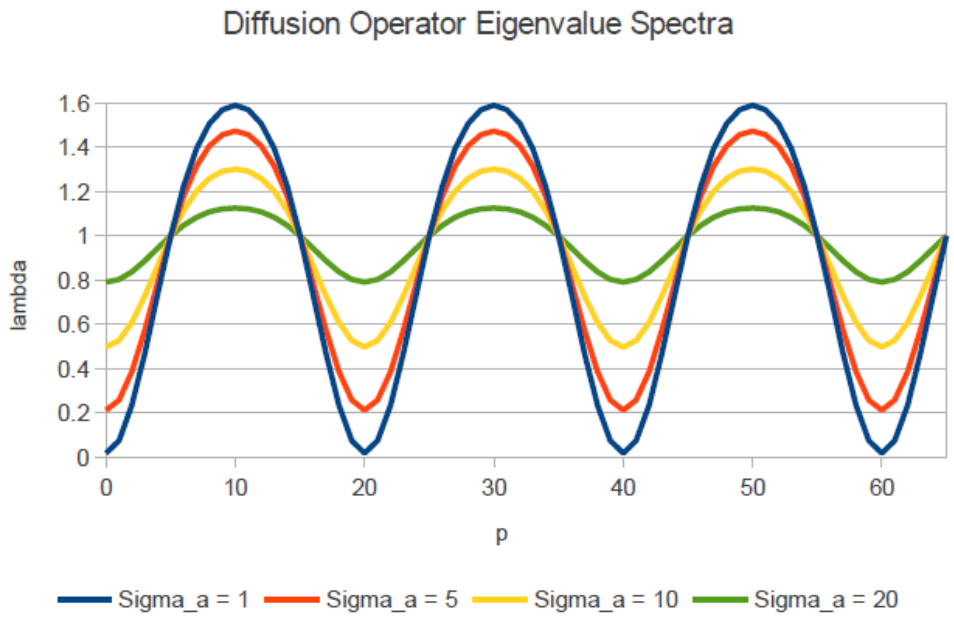

Figure 5.3: **Eigenvalue spectra for the diffusion equation.**

5.2.2 Neumann Series Convergence

As outlined in §2.2, the adjoint Monte Carlo method is effectively an approximation to a stationary method. In the adjoint Neumann-Ulam method, k iterations, equivalent to k applications of the iteration matrix, are approximated by a random walk of average length k to yield the summation in Eq (2.39) (Dimov et al., 1998; Danilov et al., 2000). This random walk length, or the number of transitions before the termination of a history (either by the weight cutoff, absorption, or exiting the global domain) is therefore approximately the number of stationary iterations required to converge to the specified tolerance. In the case of the adjoint Neumann-Ulam method, no such tolerance exists, however, we have specified a weight cutoff, W_c, that determines when low-weight histories will be prematurely terminated as their contributions are deemed minute. After k iterations, a stationary method is terminated as the error has reached some fraction, ϵ, of the initial error:

$$||\mathbf{e}^k||_2 = \epsilon||\mathbf{e}^0||_2 \,. \tag{5.11}$$

Per Eq (A.10), we see that this fraction is equivalent to $\epsilon = \rho(\mathbf{H})^k$. For the adjoint Neumann-Ulam method, if we take this fraction to be the weight cutoff, a measure of how accurately the contributions of a particular history to the solution are tallied, we then have the following relationship for k:

$$k = \frac{\log(W_c)}{\log(\rho(\mathbf{H}))} \,. \tag{5.12}$$

This then gives us a means to estimate the length of the random walks that will be generated from a particular linear operator based on the eigenvalues of its iteration matrix (independent of the linear operator splitting chosen) and based on the weight cutoff parameter used in the Neumann-Ulam method.

5.2.3 Domain Leakage Approximations

In a domain decomposed situation, not all histories will remain within the domain they started in and must instead be communicated. This communication, expected to be expensive, was analyzed by Siegel and colleagues for idealized, load balanced situations for full nuclear reactor core Monte Carlo neutral particle simulations (Siegel et al., 2012a). To quantify the number of particles that leak out of the local domain

they define a leakage fraction, Λ, as:

$$\Lambda = \frac{average\ \#\ of\ particles\ leaving\ local\ domain}{total\ of\ \#\ of\ particles\ starting\ in\ local\ domain} . \tag{5.13}$$

For their studies, Siegel and colleagues assumed that the value of Λ was dependent on the total cross section of the system via the Wigner rational approximation. Outlined more thoroughly by Hwang's chapter in (Azmy and Sartori, 2010), we will use both the Wigner rational approximation and the mean chord approximation as a means to estimate the leakage fraction.

In the case of domain decomposed linear systems, we can use diffusion theory to estimate the optical thickness of a domain in the decomposition and the corresponding leakage fraction in terms of properties of the linear operator and the discretization. To begin we must first calculate the mean distance a Monte Carlo history will move in the grid by computing the mean squared distance of its movement along the chord of length l defined across the domain. After a single transition a history will have moved a mean squared distance of:

$$\langle \bar{r_1^2} \rangle = (n_s h)^2 , \tag{5.14}$$

where h is the size of the discrete grid elements along the chord and n_s is the number of grid elements a history will move on average every transition. For our diffusion model problem, n_s would equate to the expected number of states in the i (or j as the problem is symmetric) direction that a history will move in a single transition and is dependent on the stencil used for the discretization. After k transitions in the random walk, the history will have moved a mean squared distance of:

$$\langle \bar{r_k^2} \rangle = k(n_s h)^2 . \tag{5.15}$$

If our chord is of length l and there are n_i grid elements (or states to which a history may transition) along that chord, then $h = l/n_i$ giving:

$$\langle \bar{r_k^2} \rangle = k\left(\frac{n_s l}{n_i}\right)^2 . \tag{5.16}$$

From diffusion theory, we expect the average number of interactions along the chord to be:

$$\tau = \frac{l}{2d\sqrt{\langle \bar{r_k^2} \rangle}} , \tag{5.17}$$

where d is the dimensionality of the problem and $\sqrt{\langle \bar{r}_k^2 \rangle}$ is effectively the mean free path of the Monte Carlo history in the domain. We can readily interpret τ to be the *effective optical thickness* of a domain of length l. Inserting Eq (5.16) we arrive at:

$$\tau = \frac{n_i}{2dn_s\sqrt{k}}\,, \tag{5.18}$$

which if expanded with Eq (5.12) gives us the final relation for the effective optical thickness:

$$\tau = \frac{n_i}{2dn_s}\sqrt{\frac{\log(\rho(\mathbf{H}))}{\log(W_c)}}\,. \tag{5.19}$$

For optically thin domains, we expect that most histories will be communicated, while optically thick domains will leak the fraction of histories that did not interact within. Using the optical thickness defined in Eq (5.19), we can then complete the leakage approximations by defining the bounds of $\tau \to 0, \Lambda \to 1$ and $\tau \to \infty, \Lambda \to \tau^{-1}$. With these bounds we can then define the leakage fraction out of a domain for the adjoint Neumann-Ulam method using the Wigner rational approximation:

$$\Lambda = \frac{1}{1+\tau}\,, \tag{5.20}$$

and using the mean-chord approximation:

$$\Lambda = \frac{1-e^{-\tau}}{\tau}\,. \tag{5.21}$$

Here, the leakage fraction is explicitly bound to the eigenvalues of the iteration matrix, the size of the domain, the content of the discretization stencil, and the weight cutoff selected to terminate low weight histories.

5.2.4 Numerical Experiments

To test the relationships developed by the spectral analysis, we form two simple numerical experiments using the diffusion model problem: one to measure the length of the random walks as a function of the iteration matrix eigenvalues, and one to measure the domain leakage fraction as a function of the iteration matrix eigenvalues and the discretization properties. Before doing this, we verify our computation of the spectral radius of the iteration matrix by numerically computing the largest eigenvalue of the diffusion operator using an iterative eigenvalue solver. For this verification, a 100×100 square grid with $h = 0.01$, $h = 0.1$, and $h = 1.0$ and the absorption cross

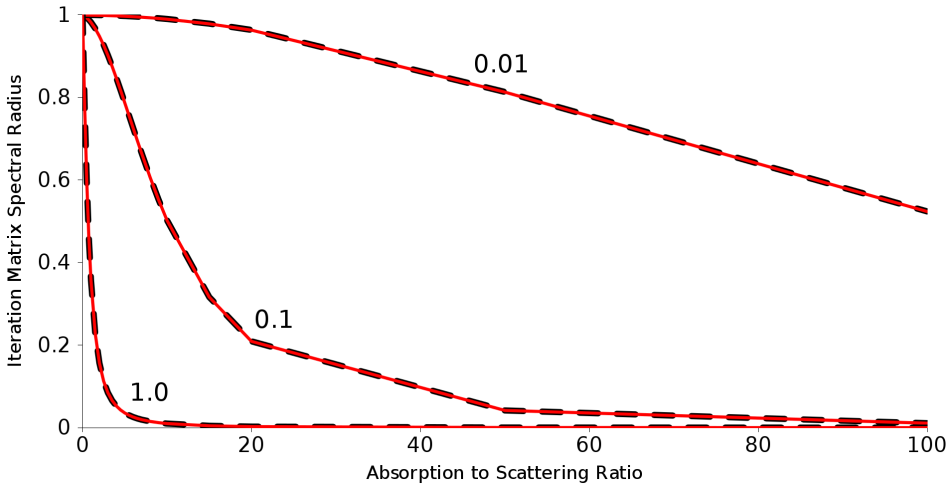

Figure 5.4: **Measured and analytic preconditioned diffusion operator spectral radius as a function of the absorption cross section to scattering cross section ratio.** *Values of* $h = 0.01$*,* $h = 0.1$*, and* $h = 1.0$ *were used. The red data was computed numerically by an eigensolver while the black dashed data was generated by Eq (5.10).*

varied from 0 to 100 while the scattering cross section was fixed at unity. Figure 5.4 gives the measured spectral radius of the iteration matrix and the computed spectral radius for the preconditioned diffusion operator using Eq (5.10) as function of the absorption to scattering ratio (Σ_a/Σ_s). Excellent agreement was observed between the analytic and numerical results with all data points computed within the tolerance of the iterative eigenvalue solver.

5.2.4.1 Random Walk Length

With the eigenvalue derivations verified, we can go about setting up an experiment to measure the length of the random walks generated by the adjoint Neumann-Ulam solver. To do this, we again use a 100×100 square grid with $h = 0.1$ and the absorption cross varied from 0 to 100 while the scattering cross section was fixed at unity. Three weight cutoff values of 1×10^{-2}, 1×10^{-4}, and 1×10^{-8} were used with 10,000 histories generated by a point source of strength 1 in the center of the domain. For each of the histories, the number of transitions made was tallied to provide an effective value of k for each history. This value was then averaged over all histories to get a measured value of k for the particular operator. On the left, Figure 5.5 presents these measurements as well as the analytic result computed by Eq (5.12) as a function of the iteration matrix spectral radius, $\rho(\mathbf{H})$. On the right, Figure 5.5

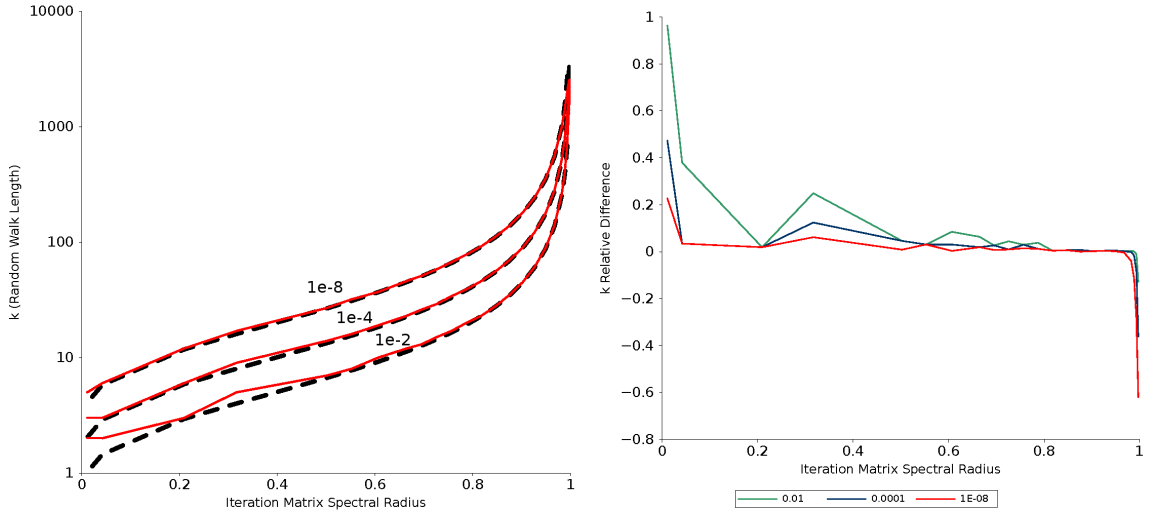

Figure 5.5: **Measured and analytic random walk length as a function of the iteration matrix spectral radius.** *The weight cutoff was varied with* 1×10^{-2}, 1×10^{-4}, *and* 1×10^{-8}. *In the left plot, the red data was computed numerically by an adjoint Neumann-Ulam implementation while the black dashed data was generated by Eq (5.12). In the right plot, the relative difference between the predicted and measured results is presented for each weight cutoff.*

gives the relative difference between the predicted and observed results. We note good qualitative agreement between the measured and analytic results. However, we observe a larger relative difference for both long and short random walks.

5.2.4.2 Domain Leakage

Finally, we seek to measure the leakage from a domain in a domain decomposed Monte Carlo calculation and assess the quality of our analytic relation for the optical thickness of a domain and the associated leakage approximations. For this experiment, a square grid with $h = 0.1$ was decomposed into 9 square domains, 3 in each cardinal direction with measurements occurring in the central domain without boundary grid points. For the cross sections, the absorption cross section was varied from 1 to 100 while the scattering cross section was set to zero to create a purely absorbing environment with weight cutoff of 1×10^{-4}. The optical thickness of these domains will vary as a function of the absorption cross section if the other parameters are fixed.

To compute the optical thickness, along with the spectral radius as given by Eq (5.10), we also need the parameters n_i and n_s which respectively describe the typical domain length and the average number of states moved along that typical

length per history transition. For our grid above, the domains are varied in size with 50×50, 100×100, and 200×200 cells giving $n_i = 50$, $n_i = 100$, and $n_i = 200$ grid points or states along the typical length of the domain respectively. Looking at the Laplacian stencil in Eq (F.4), we see that all history transitions will only move a single state in either the i or j directions due to the symmetry of the problem. Furthermore, if we choose the i direction, not all states we will transition to will move the history in that direction. Therefore, we look to the definition of the iteration matrix in Eq (5.7) and the definition of the adjoint probability matrix in Eq (2.44) to estimate the n_s parameter. For a particular transition starting at state (i, j), 6 of the 8 possible new states in the stencil move the history in i direction with relative coefficients of 4 for moving in the $(\pm i, 0)$ direction and of 1 for moving in the $(\pm i, \pm j)$. These coefficients dictate the frequency those states are visited relative to the others. For those 6 states we can visit along the typical length, their sum is 12 out of the total 20 for the coefficients for all possible states with their ratio giving $n_s = \frac{3}{5}$.

To compute the leakage fraction numerically, 3×10^5 histories were sampled from a uniform source of strength unity over the global domain. At the start of a stage of histories, the number of histories starting in the center domain was computed and as the stage progressed, the number of histories that exited that domain was tallied with the ratio of the two numbers providing a numerical measure for the leakage fraction. Figure 5.6 gives the domain leakage measurements for the domain in the center of the global grid as well as the analytic result computed by Eqs (5.20) and (5.21) as a function of the iteration matrix spectral radius. Again, we note good qualitative agreement between the measured and analytic quantities but we begin to see the limits of the leakage approximations.

To compare the quality of the two approximations, the absolute difference between the computed leakage fraction and that generated by the Wigner rational and mean chord approximations is plotted in Figure 5.7 for all domain sizes tested. From these difference results, the mean chord approximation is shown to have a lower difference for ill-conditioned systems as compared to the Wigner approximation while the Wigner approximation produces less difference for more well-conditioned systems. We also note that for the optically thick domains, the difference is likely corresponded to that observed in Figure 5.5 for the k parameter while the large relative difference in k for optically thin domains does not affect the approximation significantly. In general, the mean chord approximation is a better choice to estimate the leakage fraction in a domain from the adjoint Neumann-Ulam method and except for a single data point with $n_i = 50$, the mean chord approximation yielded leakage fractions within 0.05

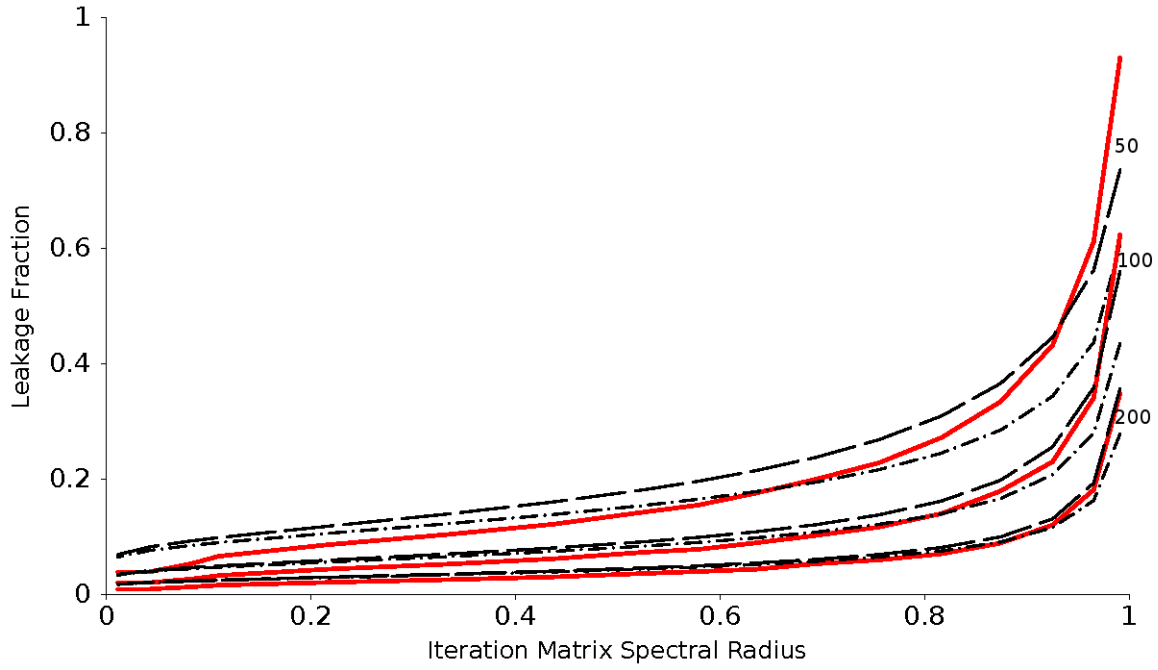

Figure 5.6: **Measured and analytic domain leakage as a function of the iteration matrix spectral radius.** *To test the behavior with respect to domain size,* $n_i = 50$*,* $n_i = 100$*,and* $n_i = 200$ *were used. The red data was computed numerically by a domain-decomposed adjoint Neumann-Ulam implementation, the black dashed data was generated by Eq (5.21) using the mean-chord approximation, and the dashed-dotted black data was generated by Eq (5.20) using the Wigner rational approximation.*

of the measured results. As the domain becomes more optically thick (with both increasing n_i and decreasing $\rho(\mathbf{H})$), the approximations are more accurate.

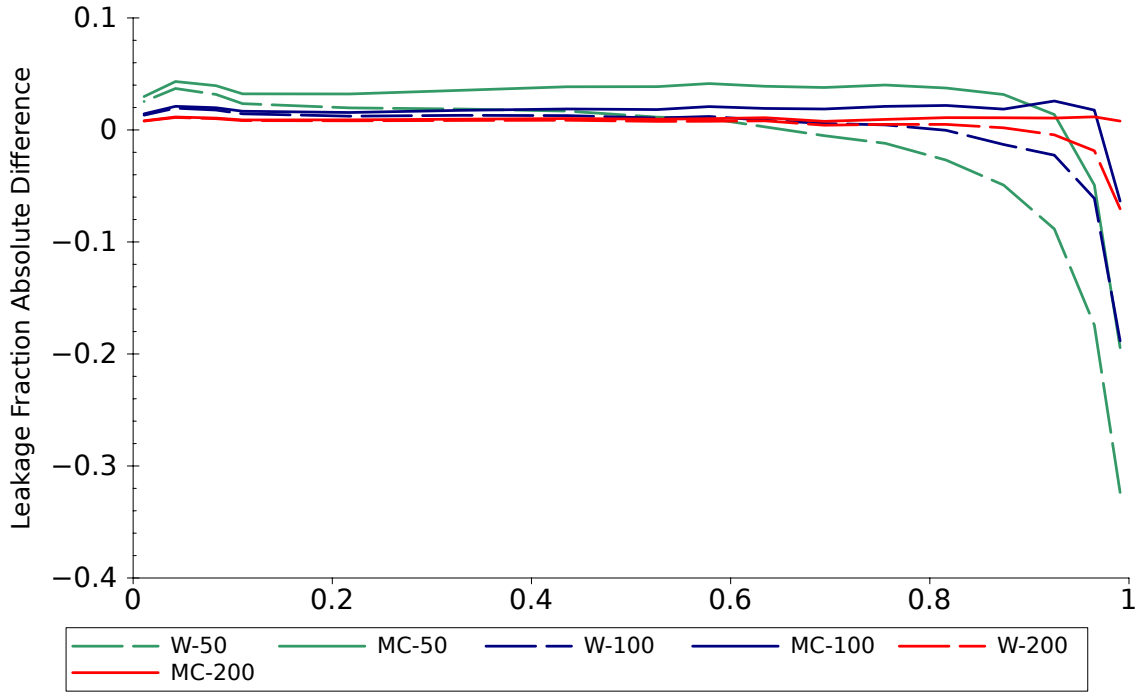

Figure 5.7: **Measured and analytic domain leakage absolute difference as a function of the iteration matrix spectral radius.** *To test the behavior with respect to domain size, $n_i = 50$ (green), $n_i = 100$ (blue), and $n_i = 200$ (red) were used. The dashed lines represent the difference using the Wigner rational approximation while the solid lines represent the difference using the mean-chord approximation.*

5.3 Domain Decomposed Neumann-Ulam Algorithm

In the context of radiation transport, in 2009 Brunner and colleagues provided a fully asynchronous domain decomposed parallel algorithm as implemented in production implicit Monte Carlo codes (Brunner and Brantley, 2009). We will adapt their algorithm and directly apply it to a parallel formulation of the Neumann-Ulam method. Direct analogs can be derived from their works by noting that the primary difference between solving a linear transport system with Monte Carlo methods and traditional fixed source Monte Carlo transport problems is the content of the Markov chains that are generated.

The transitions represented by these chains are bound by probabilities and weights and are initiated by the sampling of a discrete source. In the context of transport problems, those transitions represent events such as particle scattering and absorption with probabilities that are determined by physical data in the form of cross sections. For stochastic matrix inversion, those transitions represent moving between the equations of the linear system (and therefore the components of phase space which they represent) and their probabilities are defined by the coefficients of those equations. Ultimately, we tally the contributions to generate expectation values in the desired states as we progress through the chains. Therefore, parallel methods for Monte Carlo radiation transport can be abstracted and we can use those concepts that apply to matrix inversion methods as an initial means of developing a parallel Neumann-Ulam-type solver.

In this section Brunner and Brantley's fully asynchronous algorithm, which was effectively implemented verbatim for this work, is presented along with its application to the Neumann-Ulam method. For considerably more detail, the algorithm presented can be found in (Brunner and Brantley, 2009). In their work they identify two data sets that are required to be communicated: the sharing of particles that are transported from one domain to another and therefore from one processor to another and a global communication that signals if particle transport has been completed on all processors. Both of these communication sequences will be addressed at a high level along with how the Monte Carlo data structures they require are constructed in parallel.

5.3.1 Parallel Transport Domain and Source

To utilize a parallel transport algorithm, we must generate the required data with the correct parallel decomposition. For the Neumann-Ulam method, the transport domain consists of all states in the system that are local, and the probabilities and weights for all state transitions possible in the local domain. At the core of this representation is the Neumann-Ulam decomposition of the linear operator as given by Eq 2.13. Given an input decomposition for the linear operator, by definition the Neumann-Ulam decomposition will have the same parallel decomposition. In addition, any modification made to the linear operator through preconditioning or relaxation parameters will also modify the Neumann-Ulam decomposition. All possible transitions are described by the local graph of the sparse input matrix. In addition, any left preconditioning or relaxation parameters will modify the right hand side of the linear system and therefore the fixed source in the Monte Carlo calculation. This source vector will have the same parallel decomposition as the input matrix and therefore all of the birth states for the histories will exist for at least a single transition event within the local domain.

Compared to the serial construction of the Neumann-Ulam decomposition, data for all states that is required to be on process must be collected in parallel. For the pure domain decomposition case, given an input parallel decomposition for the Neumann-Ulam decomposition, for all local states m we require access to P_{mn} and W_{mn} (or P^T_{mn} and W^T_{mn}) for the adjoint method for all possible states n. Given by Figure 5.8, consider the adjacency graph of a single state in the neutron diffusion matrix for the model problem presented in Appendix F.

In this example, we currently reside in state m of the system, directly correlating to physical location (i, j) in the mesh (the center node). The discretization stencil of the diffusion problem dictates the structure of this graph and the states to which a history may transition. If we play the Monte Carlo game to move to a new state n, then we may move to any of the nodes in this graph, including the node at which we started. If grid point (i, j) in this example is on the right of the local domain boundary and we transition to the right to state n corresponding to mesh point $(i + 1, j)$, we are now in a state that is owned by the adjacent domain. We must then gather the data in the Neumann-Ulam decomposition from the neighboring process that owns grid point $(i + 1, j)$. Doing this provides us with the data required by the estimators permitting P_{mn} and W_{mn} to be computed locally for this particular boundary transition. In addition, we also collect the identification number for the

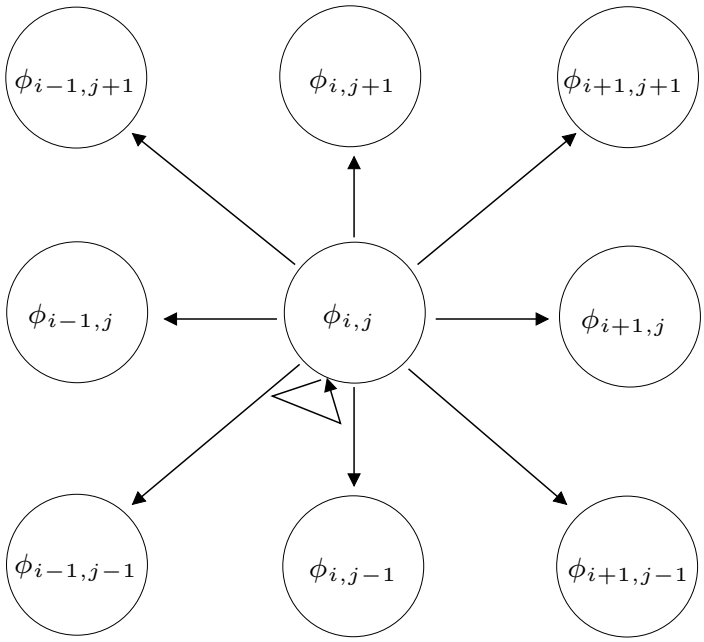

Figure 5.8: **Neutron diffusion equation state adjacency graph.** *The structure of the graph comes from the discretization of the Laplacian operator describing the diffusion physics. Adjacent boundary states are collected by traversing the graph one step for all local boundary states and collecting the equations for those states that are not local. For a given transition in the random walk for the diffusion problem, a history at mesh point (i, j) may transition to all adjacent mesh points including itself, corresponding to a global state transition from m to n in the linear system.*

process that owns this state such that we have an address to send all histories that leave the local domain boundary for this state. For the parallel source, these adjacent states are not required as histories may only be born in states in the local domain. Once the local Neumann-Ulam decomposition has been generated with the proper data collected from adjacent domains, Monte Carlo transport may proceed in parallel.

5.3.2 Domain Decomposed Algorithm

Here we present in detail Brunner and Brantley's 2009 algorithm in detail and discuss how it is adapted to parallelize the Neumann Ulam method. Presented in Algorithm 5.1, the top level sequence performs the task of managing history transport through the local domain, communication of histories to adjacent domains, and the completion of transport. For each of these specific tasks, additional algorithms, shown in bold in Algorithm 5.1 (e.g. **LocalHistoryTransport()**), are presented for additional detail in the same manner as Brunner and Brantley.

Algorithm 5.1 Parallel Neumann-Ulam Algorithm

```
1: get list of neighbor processors        ▷ Each neighbor owns an adjacent subdomain
2: for all neighbors do
3:     post nonblocking receive for maximum history buffer size
4:     allocate history buffer
5: end for
6: historiesCompleted = 0                              ▷ local+child finished histories
7: localProcessed = 0          ▷ local trajectories computed (not necessarily finished)
8: calculate parent and children processor ID numbers in binary tree
9: for all child processes do
10:    post nonblocking receive for historiesCompleted tally
11: end for
12: post nonblocking receive for stop message from parent
13: while stop flag not set do
14:    if any local histories in source or stack then
15:        LocalHistoryTransport()
16:        ++localProcessed
17:    end if
18:    if message check period == localProcessed || no local histories then
19:        ProcessMessages()
20:        localProcessed = 0
21:    end if
22:    if no local histories then
23:        ControlTermination()
24:    end if
25: end while
26: cancel outstanding nonblocking requests
27: free all history buffers
```

Successful execution of this algorithm requires construction of the parallel Neumann-Ulam decomposition as described in the preceding section. This data is used to begin Algorithm 5.1 where lines 1-5 use the collected list of neighboring processors generated

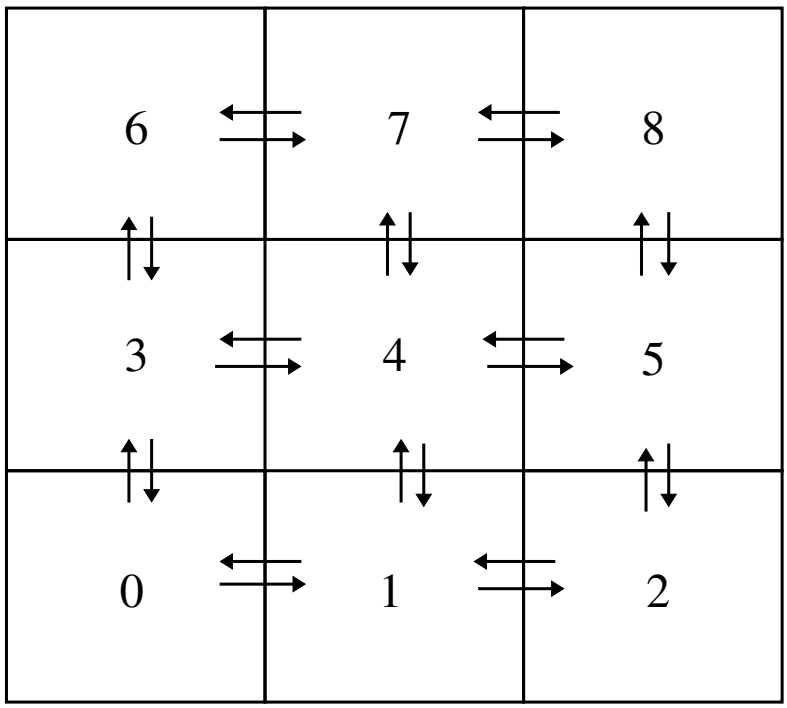

Figure 5.9: **Nearest neighbor history communication sequence.** *Each subdomain in the system has a set of nearest neighbors determined by the parallel adjacency graph of the input matrix. The subdomains are indexed by an integer.*

while building the Neumann-Ulam decomposition to setup the set of asynchronous messages required for domain-to-domain communication. Again, consider this communication pattern for the 9 subdomain example given by Figure 5.9. In this pattern, each boundary domain has two neighbors and the center domain four neighbors with which they will communicate parallel histories and this communication goes both ways as represented by the adjacent arrows in the figure. For each set of neighbors, a non-blocking send and receive is required with data buffers allocated with a user-defined size prepared for incoming and outgoing histories. This nonblocking structure is critical to the performance of the algorithm in that it permits local history transport to continue while new histories to transport are being collected in the buffers. When a given process is ready to do more work, it can check these data buffers for incoming histories requiring further transport. In this way there is a maximum amount of overlap between communication and transport of histories.

Once the nearest-neighbor communication sequence has been prepared, the completion of transport sequence is readied in lines 6 and 8-12 in Algorithm 5.1. In these lines we are setting up an asynchronous binary communication tree as presented for the same

9 subdomain example in Figure 5.3.2. In this communication pattern, each process has one parent process (except for the MASTER process 0) to which it will nonblocking send the number of histories that terminated their transport procedure by weight cutoff in the local domain. Equivalently, each process has up to two child processes from which it will receive their completed number of histories with a nonblocking receive operation. Setting up a tree in this manner lets the completed history tally (*historiesCompleted* in the algorithm) be updated incrementally and funneled to the root process. Because we are solving fixed source problems with the Neumann-Ulam algorithm without any type of variance reduction that may generate more histories that we started with, once the root process completion sum tallies to the number of histories in the source, transport is complete. Once this occurs, the root process nonblocking sends a process to its children and each process nonblocking receives a stop message from its parent as shown in Figure 5.3.2. The stop message is then propagated up the tree in the same manner.

With these communication structures prepared, we can now enter the main transport loop at line 13 of Algorithm 5.1. In this loop, a set of mutually exclusive tasks enabled by the fully asynchronous communication patterns are executed until the stop signal is received from the parent process in the binary tree. In this case, mutually exclusivity permits all local operations to occur independently of other processes in the system and their current state. For each process, there are two data structures from which histories may be obtained for the transport procedure. The first is the local source and the second is a LIFO (last in first out) stack of histories transported to the local domain from the adjacent domain. In lines 14-17, if histories exist in either of these data structures, then they are transported through the local domain using Algorithm 5.2. If the history is terminated by weight cutoff the *historiesCompleted* tally is updated. If it hits the boundary of the domain it is added to the outgoing history buffer for the neighboring domain and if the buffer is full, it is sent to the neighboring domain with a nonblocking operation and the memory associated with that buffer reallocated. In all instances that Algorithm 5.2 is executed, *localProcessed* is incremented to account for histories that have been processed locally.

Continuing in the transport loop, if there are no local histories to transport or the *localProcessed* count has reached some user-defined check frequency, lines 18-21 in Algorithm 5.2 check for more histories to transport in the incoming data buffers by calling Algorithm 5.3. For each neighboring domain in the problem, if there are histories in those data buffers then they are added to the stack from processing and the nonblocking receive operation re-instantiated. In addition, the terminated histories

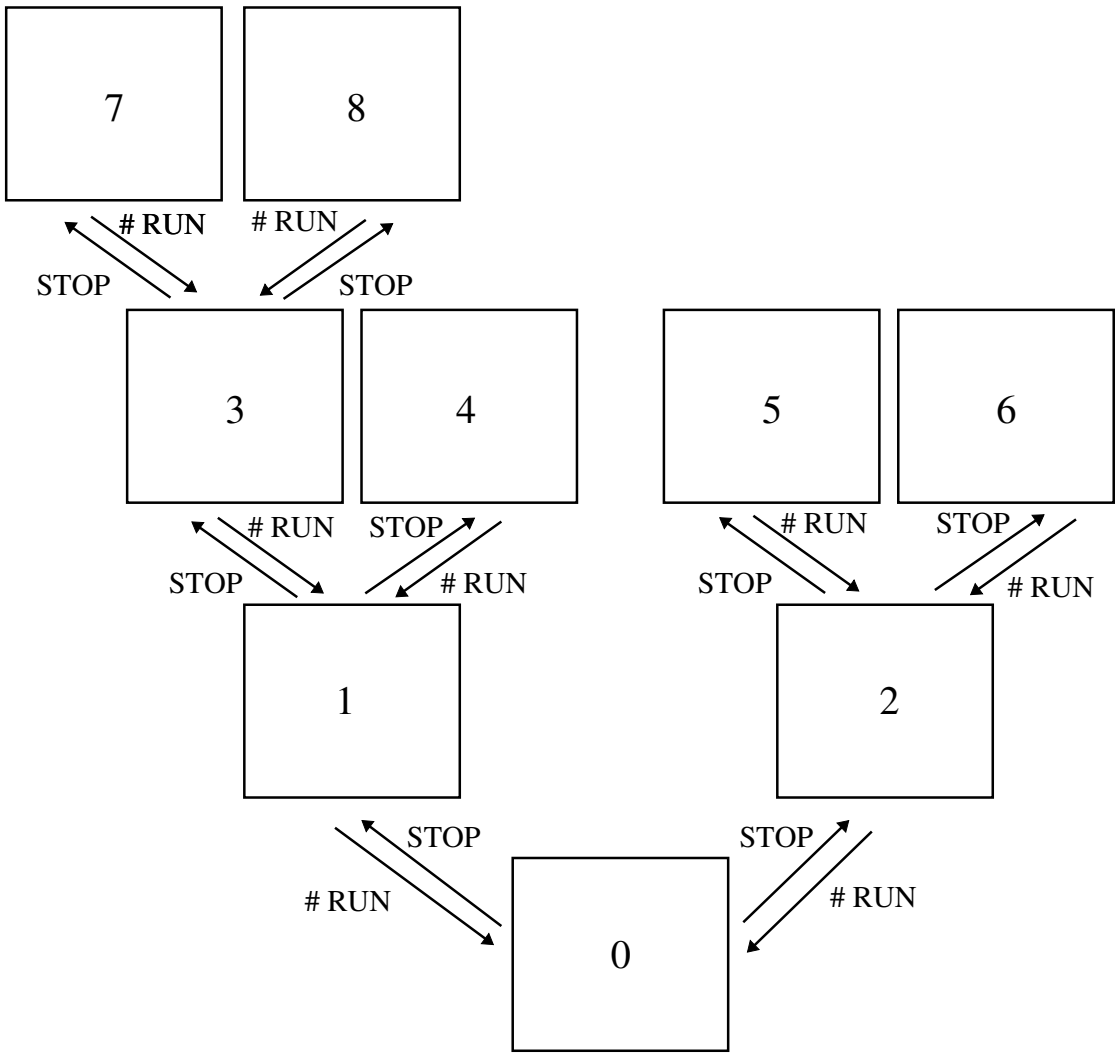

Figure 5.10: **Binary communication tree for coordinating the end of a parallel Neumann-Ulam solve.** *Each child process reports to its parents how many histories completed within its domain. When the root process sums all complete histories, it forwards the stop signal to its children which in turn forward the message up the tree. The subdomains are indexed by an integer.*

count is updated with those values received from the child processes.

Finally, the transport loop finishes in lines 22-24 of Algorithm 5.2 by calling Algorithm 5.4 if there are no local histories to transport in the stack or the source. In Algorithm 5.4, we continue to forward the terminated history tally down to the parent in the binary tree and send all history buffers to our neighbors, even if they are not full. If Algorithm 5.4 is called by the MASTER process, it checks for completion of the problem and if it is complete, forwards to stop flag onto its children as in Figure 5.3.2. If the process is not the MASTER, then we check for a stop signal from the parent

Algorithm 5.2 LocalHistoryTransport()

```
1: transport history through the domain until termination
2: if history is in neighbor boundary state then
3:     add history to neighbor buffer
4:     if neighbor buffer is full then
5:         nonblocking send history buffer to neighbor
6:         allocate new history buffer for neighbor
7:     end if
8: else
9:     if history terminated by weight cutoff then
10:        post-process history
11:        ++historiesCompleted
12:    end if
13: end if
```

Algorithm 5.3 ProcessMessages()

```
1: for all received history buffers from neighbor do
2:     unpack number of histories in buffer
3:     add the histories to the running stack
4:     repost nonblocking receive with neighbor
5: end for
6: for all historiesCompleted messages from children do
7:     historiesCompleted += message value
8:     repost nonblocking receive with child
9: end for
```

and forward to the child processes if transport is complete.

In this manner the transport loop continues until all process have received the stop signal from their parents. Older versions of this algorithm, particularly some of those presented in (Brunner et al., 2006), use a master/slave approach as given by Figure 5.3.2 for the completion of a transport stage instead of the binary tree scheme. In this approach, the root process still manages the final summation of the completion tally, it directly receives completed tally results from all other processes in the problem instead of from just its children. Although this implementation is potentially simpler to understand than the binary tree approach, the authors of (Brunner et al., 2006) observed that this type of termination pattern, even when implemented in a fully asynchronous manner, was a major scalability bottleneck. This bottleneck is due to the fact that the master process falls behind the others, creating a load imbalance even with the addition of a few integer addition operations. We will explicitly demonstrate in scaling studies how this bottleneck also occurs within a

Algorithm 5.4 ControlTermination()

```
 1: nonblocking send an partially full history buffers
 2: for all historiesCompleted messages from children do
 3:     historiesCompleted += message historiesCompleted
 4:     repost nonblocking receive with child
 5: end for
 6: if MASTER processor then
 7:     if historiesCompleted == global # of source histories then
 8:         set stop flag
 9:         for all children do
10:             nonblocking send stop message to child
11:         end for
12:     end if
13: else
14:     nonblocking send historiesCompleted to parent
15:     historiesCompleted = 0
16:     check for stop signal from parent
17:     if stop signal from parent then
18:         for all children do
19:             nonblocking send stop message to child
20:         end for
21:     end if
22: end if
```

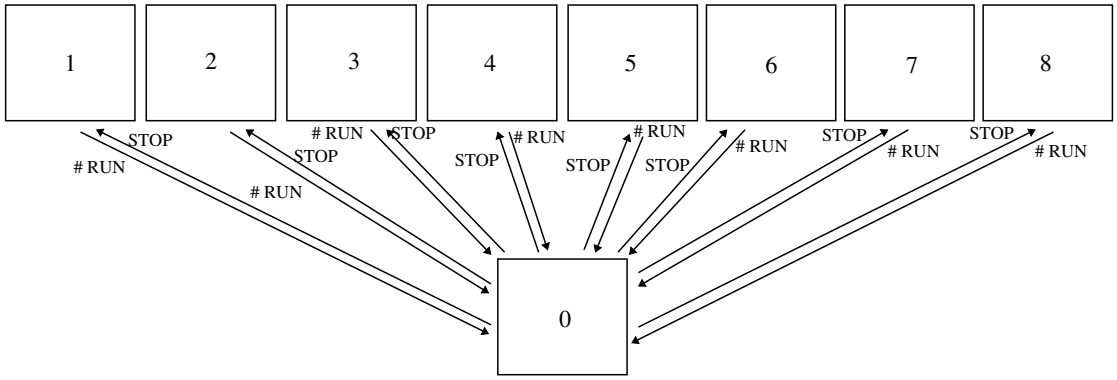

Figure 5.11: **Master/slave scheme for coordinating the end of a parallel Neumann-Ulam solve.** *Each slave process reports to the master how many histories completed within its domain. When the master process sums all complete histories, it sends the stop signal to all slave processes. The subdomains are indexed by an integer.*

Neumann-Ulam implementation of this algorithm thus requiring the implementation of a binary communication tree.

There is an additional advantage to Brunner and Brantley's work that although not

immediately applicable to this work has the potential to provide value in the future. In addition to a robust fully asynchronous communication pattern, this algorithm may also be modified to account for situations where the total number of histories in a given stage are not known before starting transport. From a physics perspective, we might expect this for situations where perhaps an (n,2n) interaction occurs in a neutronics problem. In this case, the algorithm is modified to account for both histories terminated and created and several mechanisms are introduced to determine completion of the transport stage. For future variations of this work, certain variance reduction techniques which create histories, such as splitting, have the potential to be successfully employed as a means of accelerating the time to solution for a given problem using MCSA. The parallel Neumann-Ulam algorithm presented here may be adapted to account for these additional techniques.

5.4 Multiple-Set Overlapping-Domain Algorithm

Although the implementation presented in the previous section was observed by Brunner and Brantley to be robust and allowed for scaling to large numbers of processors, performance issues were still noted with parallel efficiency improvements needed in both the weak and strong scaling cases for unbalanced problems. These results led them to conclude that a combination of domain decomposition and domain replication could be used to solve some of these issues. In 2010, Wagner and colleagues developed the *multiple-set overlapping-domain* (MSOD) decomposition for parallel Monte Carlo applications for full-core light water reactor analysis (Wagner et al., 2010). In their work, an extension of Brunner's, their scheme employed similar parallel algorithms for particle transport but a certain amount of overlap between adjacent domains was used to decrease the number of particles leaving the local domain. In addition, Wagner utilized a level of replication of the domain such that the domain was only decomposed on $O(100)$ processors and if replicated $O(1,000)$ times potentially achieves efficient simulation on $O(100,000)$ processors, thus providing both spatial and particle parallelism.

Each collection of processors that constitutes a representation of the entire domain is referred to as a set, and within a set overlap occurs among its sub-domains. The original motivation was to decompose the domain in a way that it remained in a physical cabinet in a large distributed machine, thus reducing latency costs during communication. A multiple set scheme is also motivated by the fact that communication during particle transport only occurs within a set, limiting communications during the transport procedure to a group of $O(100)$ processors, a number that was shown to have excellent parallel efficiencies in Brunner's work and therefore will scale well in this algorithm. The overlapping domains within each set also demonstrated reduced communication costs. On each processor, the source is sampled in the local domain that would exist if no overlap was used while tallies can be made over the entire overlapping domain.

To demonstrate this, consider the example adapted from Mervin's work with Wagner and others in the same area (Mervin et al., 2012) and presented in Figure 5.12. In this example, 3 particle histories are presented emanating from the blue region of interest. Starting with particle A, if no domain overlap is used then the only the blue domain exists on the starting processor. Particle A is then transported through 3 other domains before the history ends, therefore requiring three communications to occur in Brunner's algorithm. If a 0.5 domain overlap is permitted as shown by the dashed line, then the starting process owns enough of the domain such that no

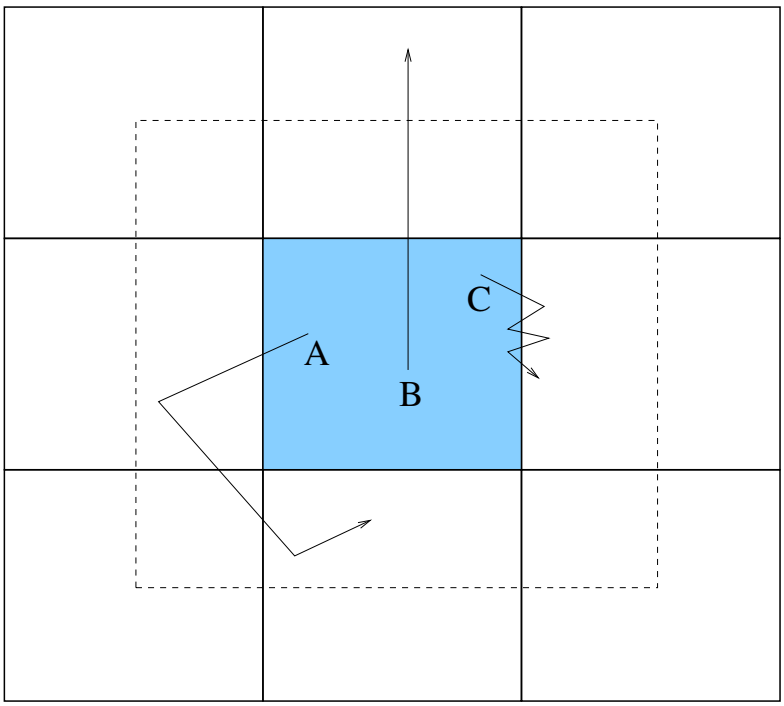

Figure 5.12: **Overlapping domain example illustrating how domain overlap can reduce communication costs.** *All particles start in the blue region of interest. The dashed line represents 0.5 domain overlap between domains.*

communications must occur in order to complete the particle A transport process. Using 0.5 domain overlap also easily eliminates cases such as that represented by the path of particle C. In this case, particle C is scattering between two adjacent domains, incurring a large latency cost for a single particle. Finally, with particle B we observe that 0.5 domain overlap will still not eliminate all communications. However, if 1 domain overlap were used, the entire geometry shown in Figure 5.12 would be contained on the source processor and therefore transport of all 3 particles without communication would occur.

Wagner and colleagues used this methodology for a 2-dimensional calculation of a pressurized water reactor core and varied the domain overlap from 0 to 3 domain overlap (a 7×7 box in the context of our example) where a domain contained an entire fuel assembly. For the fully domain decomposed case, they observed that 76.12% of all source particles leave the domain. At 1.5 domain overlap, the percentage of source particles born in the center assembly leaving the processor domain dropped to 1.05% and even further for 0.02% for the 3 domain overlap. Based on their results, we hypothesize that the overlap approach coupled with the multiple sets paradigm

that will enhance the scalability of the pure domain-decomposition algorithm for Neumann-Ulam presented in the previous section.

5.4.1 Generating the MSOD Decomposition

In § 5.3.1 we discussed how we generated the parallel transport domain from the Neumann-Ulam decomposition of a domain decomposed linear operator. We can readily adapt those data structures to account for the extra information required by an implementation of the MSOD algorithm. First, we consider the generation of overlap. Conveniently, this is identical to the neighboring states discovery problem discussed in conjunction with Figure 5.8. To generate the boundary for the local transport domain, we needed to gather all data from the immediately adjacent states in the system that did not already exist in the local domain. We solved this problem by traversing the graph of the matrix for each of the boundary states and determining which processes owned those adjacent states.

To generate overlap, we use the identical algorithm but in this case we perform as many graph level traversals as the specified amount of overlap. For example, consider the analytic relations derived in § 5.2 for the length of random walks in a Neumann-Ulam method and the amount of leakage from a domain considering a neutron diffusion problem. From these relations we might determine that, for our particular problem, if we can grow the domain by 10 additional discrete state transitions, we can reduce the number of histories that must be communicated by a significant fraction. We determine the information we must gather from neighboring domains to build the overlap by traversing the graph of the matrix outward from the boundary 10 levels. At each step, we find the data in the adjacent domain that we require and make that data local, incrementing the size of the overlap. Once the overlap has been gathered, one extra graph traversal on the boundary states is performed to get the new neighboring states and their owning processes.

Once overlap has been generated, the transport domain for a single set is complete. However, if multiple sets are to be used, an additional set of parallel procedures is required to generate the necessary data structures. Figure 5.13 gives a schematic representation of the MSOD construction process using 4 sets. For a problem where P parallel processors are available and S sets are to be used, each set is allocated P/S processors for computation. On the first P/S processors, the linear problem is generated. From the linear problem, the linear operator is used to construct the transport domain, the solution vector used to construct the tallies, and the forcing

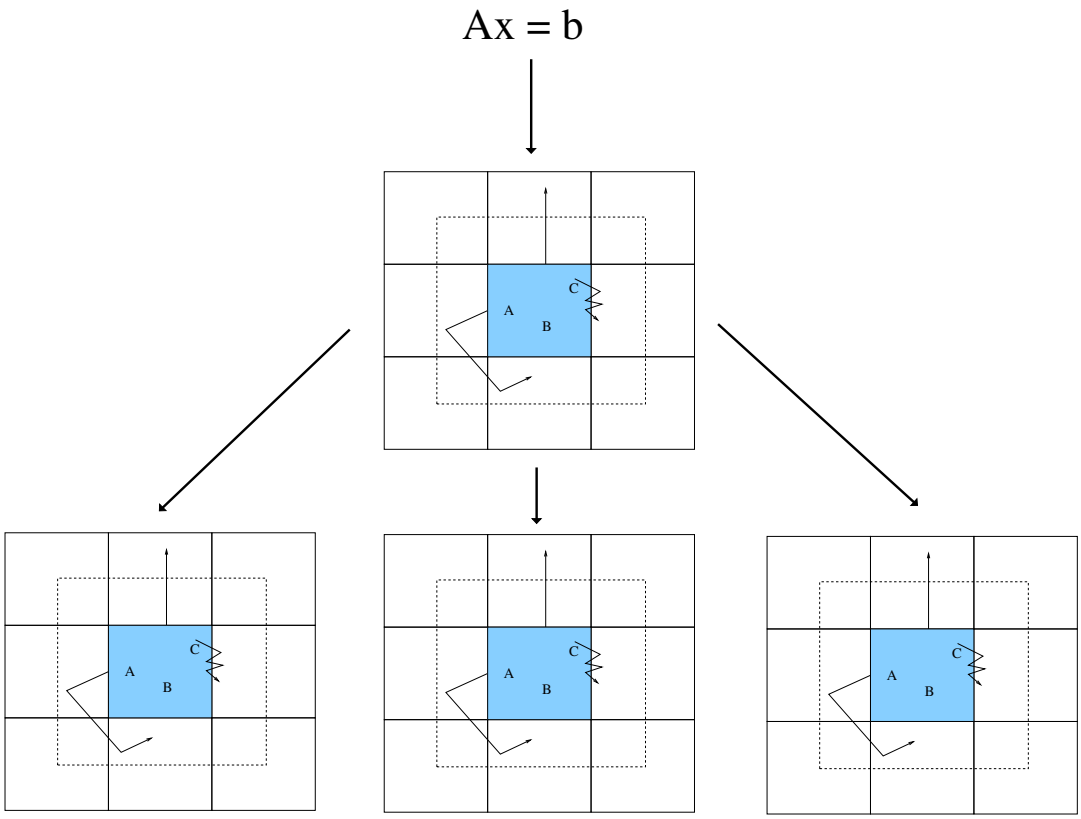

Figure 5.13: **MSOD construction for 4 sets with overlap.** *The linear system is used to construct the Monte Carlo transport domain on the primary set. The Monte Carlo data structures are then broadcast among the blocks to all sets in the system.*

term vector used to construct the fixed source. Once these Monte Carlo data structures have been generated, they are broadcast to the remaining sets in the problem as shown in Figure 5.13 such that we have S replications of the Monte Carlo problem for a single instance of the original linear problem.

5.4.2 MSOD Neumann Ulam Algorithm

With the ability to generate the transport domain within an MSOD decomposition as well as the source and tally structures, we can now define how to perform a Neumann-Ulam solve using MSOD. Given by Algorithm 5.5, we begin by constructing the Monte Carlo data structures for each set including the transport domain, the source, and the tallies. Next, in line 4 we perform the asynchronous transport algorithm presented in § 5.3 for each of the individual sets. Adding overlap to Algorithm 5.1 in this case simply modifies the set of local states and the set of neighboring states to additional states gathered during overlap generation.

Algorithm 5.5 MSOD Transport Sequence

1: build MSOD domain
2: build Monte Carlo source
3: build Monte Carlo tally
4: perform parallel Neumann-Ulam transport in each set
5: combine set tallies
6: combine block tallies

Once Monte Carlo transport is complete complete, the tallies for each individual computation must be gathered to the original set in order to build the correction vector within an MCSA sequence. To do this, we perform two parallel vector operations. First, overlap not only creates additional states in the local transport domain but also additional states in the local tally vector such that any events that occur in the local domain may be tallied in a local vector. Because of this, the tally vector in a single set will share local states with other domains and a parallel sum reduction operation is required to transform the tally into the original Neumann-Ulam parallel decomposition. Second, the tallies in individual sets must be combined and applied to the primary set in the problem that shares the processor space with the original linear problem. For this operation, we employ the concept of blocks as a subdivision of parallel space complementary to the concept of sets.

Consider the schematic representation of the final parallel reduction presented in Figure 5.14. For this operation, the tally vectors in each set must be summed together and applied to the primary set. As each set is an exact replica of the others, the parallel decomposition of the tally vector is the same within all sets. We can take advantage of this by building a processor space that encapsulates each identical domain in each set. We can then use this processor space for the reduction. For the schematic in Figure 5.14, the upper-left domain in each set creates a single block,

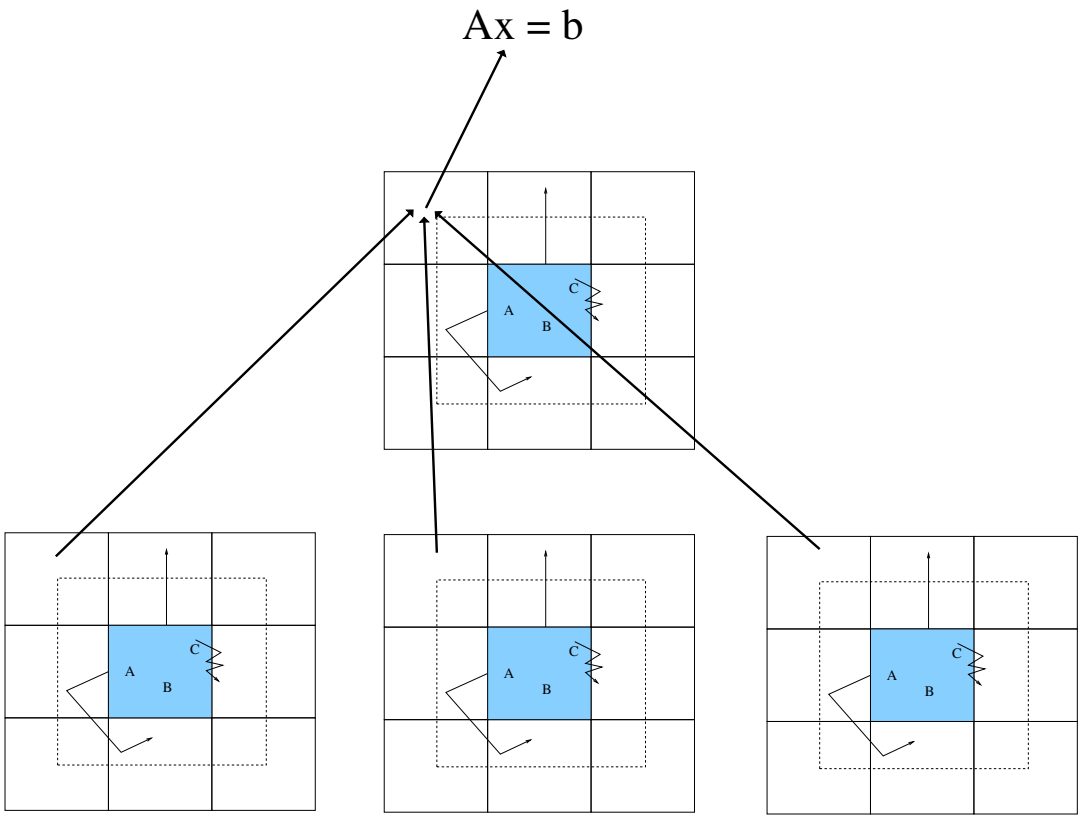

Figure 5.14: **MSOD tally reduction over blocks.** *The tally computed independently in each set is reduced across the blocks to the primary set. The linear system vector is updated on the primary set. In this example the tally vector in the block containing upper-left subdomains is combined. Identical reduction operations occur for all other blocks simultaneously.*

giving 9 total blocks for this example problem. The processor space that encapsulates each of the upper-left domains is then used for the parallel reduction operation. This reduction will happen in a mutually exclusive manner for each of the blocks in the system. One benefit of this approach is that this reduction occurs only over a subset of the processors in the problem, providing better scalability than a reduction operation over all processors in the problem.

5.5 Parallel MCSA

With the parallel adjoint Neumann-Ulam solver implementation described above, the parallel implementation of an MCSA iteration is trivial. Recall the MCSA iteration procedure presented again here for clarity:

$$\mathbf{x}^{k+1/2} = \mathbf{x}^{k} + \mathbf{r}^{k} \,, \tag{5.22a}$$

$$\mathbf{r}^{k+1/2} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1/2} \,, \tag{5.22b}$$

$$\mathbf{A}\delta\mathbf{x}^{k+1/2} = \mathbf{r}^{k+1/2} \,, \tag{5.22c}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta\mathbf{x}^{k+1/2} \,, \tag{5.22d}$$

$$\mathbf{r}^{k+1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{k+1} \,. \tag{5.22e}$$

In §A.3 we discuss parallel matrix and vector operations as utilized in conventional projection methods. We utilize these here for the parallel MCSA implementation for the computations required in Eqs (5.22a), (5.22b), (5.22d), and (5.22e). In the first step, a parallel matrix-vector multiply is used to apply the split operator to the previous iterate's solution. A parallel vector update is then performed with the source vector to arrive at the initial iteration guess. In the next step, the residual is computed by the same operations where now the operator is applied to the solution guess with a parallel matrix-vector multiply and then a parallel vector update with the source vector is performed. Once the correction is computed with a parallel adjoint Neumann-Ulam solve, this correction is applied to the guess with a parallel vector update to get the new iteration solution. Additionally, as given by Eq (2.66), 2 parallel vector reductions will be required to check the stopping criteria: one initially to compute the infinity norm of the source vector, and another at every iteration to compute the infinity norm of the residual vector.

To parallelize the solution of Eq (5.22c), we apply the MSOD Neumann-Ulam algorithm presented in the previous section. The problem is potentially replicated with overlap and the Monte Carlo procedure returns a correction vector, $\delta\mathbf{x}$, with the same parallel decomposition as the input linear problem. As was the case for generating the replications required for a multiple set implementation, the explicit matrix and vector operations required for the rest of an MCSA iteration occur only on the subset of processors containing the primary set. Only the Monte Carlo data structures exist on the rest of the processors in the problem as replication of the other operations does not generate any more information that could be used to accelerate the time to solution.

5.6 Parallel MCSA Verification

Before we consider the parallel performance of the MCSA algorithm, we must first verify that the algorithm generates the correct solution in parallel with respect to reference solutions provided by production Krylov methods. To verify correctness, we use solutions to the neutron diffusion system presented in Appendix F. As the reference computation, serial results will be used from the conjugate gradient solver in Belos, a linear solvers package in the Trilinos scientific computing libraries (Heroux et al., 2005). In identical fashion to the verification for the SP_N equations performed in § 3.6, we will solve the neutron diffusion problem in parallel using multiple cores on a desktop machine. For each parallel solution, then the solution at each grid point in the system will be compared to the solution at the same grid point in the serial reference case with the minimum and maximum absolute differences of these values reported.

GMRES and conjugate gradient solvers from the Belos package were used on 4 cores as a means of both assuring the Krylov solvers produce the same results on multiple cores and as a means of providing an acceptable range of minimum and maximum values in the comparison. For MCSA, a range of MSOD parameters will be used such that problems with multiple sets and multiple blocks may be tested along with overlap. For all MCSA calculations, a single history was used for every DOF in the problem to compute the correction vector. For all solvers, the flux was converged to a tolerance of 1×10^{-8} and therefore if the comparisons agree within this tolerance the solutions will be deemed correct. Table 5.1 gives the parameters for the verification problem and Table 5.3 gives the results of the calculations. Solver parameters used for MCSA are given in Table 5.2.

From the data we see that for all solvers, the parallel results agree with the reference serial results within the solution tolerance for all DOF in the system. In addition, we do not notice any systematic differences in the results as a function of number of blocks, number of sets, or amount of overlap. In addition, the parallel GMRES results have maximum absolute differences of the same magnitude as all MCSA results and a larger minimum absolute difference than all MCSA results by three orders of magnitude. Based on these results, MCSA parallelized with the MSOD algorithm, and in particular the implementation used here, generates the correct results with respect to the serial conjugate gradient reference computation.

Parameter	Value
dx	0.1
dy	0.1
N_x	400
N_y	400
N	160,000
Σ_a	5.0
Σ_s	1.0
Boundary Type	Vacuum
Linear Solver Tolerance	1×10^{-8}

Table 5.1: **Parallel MCSA verification model problem parameters.** *The neutron diffusion equation in 2 dimensions is used for the model problem.*

Parameter	Value
Histories	1 per DOF
Weight Cutoff	1×10^{-2}
Fixed Point	Richardson
Estimator	Collision

Table 5.2: **Parallel MCSA verification solver parameters.**

Solver	Blocks	Sets	Cores	Overlap	Min Difference	Max Difference
Conjugate Gradient	-	-	4	-	0.0	1.11022×10^{-16}
GMRES	-	-	4	-	5.64215×10^{-13}	3.83348×10^{-9}
MCSA	2	1	2	0	8.60423×10^{-16}	7.6806×10^{-9}
MCSA	3	1	3	0	6.41154×10^{-15}	7.65397×10^{-9}
MCSA	4	1	4	0	2.66454×10^{-15}	7.62623×10^{-9}
MCSA	2	2	4	0	1.52656×10^{-15}	7.57108×10^{-9}
MCSA	1	4	4	0	5.55112×10^{-16}	7.61631×10^{-9}
MCSA	8	1	8	0	6.13398×10^{-15}	7.51744×10^{-9}
MCSA	4	2	8	0	1.03251×10^{-14}	7.62601×10^{-9}
MCSA	2	4	8	0	5.13478×10^{-15}	7.60777×10^{-9}
MCSA	1	8	8	0	1.80411×10^{-15}	7.60651×10^{-9}
MCSA	2	1	2	5	2.47025×10^{-15}	7.58882×10^{-9}
MCSA	3	1	3	5	2.44249×10^{-15}	7.6326×10^{-9}
MCSA	4	1	4	5	7.49401×10^{-16}	7.49014×10^{-9}
MCSA	2	2	4	5	1.19349×10^{-15}	7.61814×10^{-9}
MCSA	1	4	4	5	5.55112×10^{-16}	7.61631×10^{-9}
MCSA	8	1	8	5	2.05391×10^{-15}	7.63955×10^{-9}
MCSA	4	2	8	5	9.57567×10^{-15}	7.61313×10^{-9}
MCSA	2	4	8	5	5.7454×10^{-15}	7.60767×10^{-9}
MCSA	1	8	8	5	1.80411×10^{-15}	7.60651×10^{-9}
MCSA	2	1	2	10	1.08247×10^{-15}	7.63231×10^{-9}
MCSA	3	1	3	10	3.27516×10^{-15}	7.60211×10^{-9}
MCSA	4	1	4	10	4.996×10^{-16}	7.64756×10^{-9}
MCSA	2	2	4	10	4.13558×10^{-15}	7.60259×10^{-9}
MCSA	1	4	4	10	5.55112×10^{-16}	7.61631×10^{-9}
MCSA	8	1	8	10	9.71445×10^{-16}	7.58535×10^{-9}
MCSA	4	2	8	10	7.77156×10^{-16}	7.65978×10^{-9}
MCSA	2	4	8	10	4.16334×10^{-16}	7.63191×10^{-9}
MCSA	1	8	8	10	1.80411×10^{-15}	7.60651×10^{-9}

Table 5.3: **Parallel MCSA verification results.** *Difference values given are absolute values. The neutron diffusion equation in 2 dimensions is used for the model problem with the parameters given in Table 5.1. A serial conjugate gradient calculation is used as the reference computation and neutron fluxes are compared element-wise. Parameters used for the MCSA solver are given in Table 5.2. All solvers including the reference computation were converged to tolerance of* 1×10^{-8}.

5.7 Parallel Performance Metrics

How effectively the presented algorithms parallelize MCSA will be determined by several performance measures. For this work, these measures will come directly from Keyes' primer on the scalability of domain decomposed methods and specifically his discussion on how to measure their parallel performance (Keyes, 1999). All of these metrics are derived from two measured quantities: the number of iterations required for convergence and the total wall time required for convergence. Each of these quantities are then measured in varying parameter studies by changing the number of processes in a simulation and by varying the local or global problem size. Using the notation of Keyes, we will define *parallel efficiency* as $\eta(N, P)$ where the efficiency is a function of the global problem size, N, and the number of processes in the calculation, P. There are several types of efficiencies that will be useful in our analysis of scalability with different definitions depending on the type of parametric scaling study performed. For all efficiencies, a value of 100% is considered perfect while a value of 0% signals the worst possible performance. Often, as was observed in this work, it is the case that efficiencies above 100% are computed in certain scaling studies. For the data presented in the following sections, an explanation of each observed case of super-unitary efficiencies will be provided.

Of primary interest to the nuclear engineer is the *strong scaling* performance of an algorithm (Siegel et al., 2012a). In this type of scaling study, the global problem size is fixed and the local problem size changes with the number of processes in the calculation. In this case, as P increases, N is fixed, thus reducing the local problem size and performance by having a larger ratio of communication to computational work. This type of scaling is important because typically the global problem size is fixed due to the problem of interest. Consider the desire for a high fidelity neutronics simulation of the entire core of a nuclear reactor. In this case, the global problem is known from the geometry definition including the number of fuel assemblies and the geometry of those assemblies all the way down to the resolution of the space, angle, and energy discretizations required in order to capture physical phenomena of interest. Because the global problem size is known a priori, strong scaling is an important measure of how useful a solution technique is when larger and larger values of P are used to solve the problem. Ideally, the time to solution for the the problem will decrease linearly as a function of P, permitting a faster time to solution using the maximum available resources and thus making sure all P are positively affecting the runtime of the problem.

For strong scaling, the objective is to then decrease the runtime of a fixed size global problem as a linear function of P. Based on this, we can define the strong scaling *absolute efficiency* as:

$$\eta_{strong}(N,P) = \frac{1}{P}\frac{T(N,1)}{T(N,P)}\,, \tag{5.23}$$

where $T(N,P)$ is the wall time for a computation of global size N using P processes. Note here that this particular definition is measured with respect to a serial computation. On leadership class machines, it is often the case that a problem size large enough to effectively execute on $O(10,000)$ cores or larger is significantly larger than any problem that may be solved by a single core due to memory restrictions. Therefore, we consider absolute scaling for a base case where Q processes are used instead of a serial computation:

$$\eta_{strong}(N,P|Q) = \frac{Q}{P}\frac{T(N,Q)}{T(N,P)}\,. \tag{5.24}$$

The absolute efficiency measure presented here only considers the total time to solution. Often, that total time to solution may be changing for an iterative method due to the fact that the number of iterations required to converge may be changing as a function of P. This will often happen in cases where modifying N changes the spectral character of the problem, often making it stiffer and therefore more difficult to solve when N increases. Therefore we also consider *algorithmic efficiency* from Keyes' work which considers this potential change in iteration count:

$$\eta_{alg}(N,P|Q) = \frac{I(N,Q)}{I(N,P)}\,, \tag{5.25}$$

where $I(N,P)$ is the number of iterations required to converge a problem of global size N computed using P processes. We can then adjust the absolute efficiency using this new algorithmic efficiency in order to compute the parallel scalability of a single iteration such that algorithmic effects may be neglected. We define the *implementation efficiency* as:

$$\eta_{impl}(N,P|Q) = \frac{\eta}{\eta_{alg}}\,, \tag{5.26}$$

using the absolute efficiency in the numerator, giving an effective measure of scalability of a single iteration of the algorithm.

In addition to strong scaling, in certain cases it is useful to consider the *weak scaling* of an iterative method. In this case, the ratio N/P is fixed such that the local problem size remains the same for all values of P. In this case, the objective is to

maintain a constant wall time while both N and P grow as a fixed ratio. In this case, we define the absolute efficiency as:

$$\eta_{weak}(M|N,P|Q) = \frac{T(M,Q)}{T(N,P)}\,, \tag{5.27}$$

subject to the constraint $M/Q = N/P$. The scaling is perfect if the runtime is static for all P when N/P is fixed.

Often, weak scaling is of interest to the nuclear engineer due to the fact that the computational resources available are fixed and may not be able to contain the entire global problem of interest. In this case, an algorithm with good weak scaling performance permits all of the computational resources available to be used effectively to allow for the highest fidelity problem possible to be solved. In addition, for the resources available, the memory to processor ratio on a distributed machine is often fixed, giving an exercise in fixed N/P to solve the highest fidelity problem possible. Therefore, good weak scaling should not be viewed as an enhancement of time to solution for the engineer but rather permitting a higher fidelity problem to be solved within a given time constraint. For problems where increasing fidelity increases the stiffness of the system, there is often a trade-off between the absolute efficiencies computed for a given method and the degrading algorithmic efficiencies that arise when increasing the fidelity. Both time and hardware constraints should be considered when preparing a parallel calculation.

5.8 Leadership-Class Parallel Scaling Studies

Using the parallel performance metrics presented in the previous section, we will next measure performance in various scaling studies using an implementation of the parallel MCSA algorithm to assess its quality using the neutron diffusion problem presented in Appendix F. For these scaling studies, large problems at high levels of concurrency will be solved at a leadership-class computing facility. In this section, we will outline the parallel machine on which the calculations were performed and then present a series of numerical studies that identify both strengths and weaknesses of the parallel algorithm. Both pure domain decomposition and MSOD parallel performance will be explored and compared to performance results for the same problems using a set of production parallel Krylov solvers.

For the scaling results presented here, all parallel MCSA results were computed using the Monte Carlo Linear Solvers[1] (MCLS) library generated specifically for this work while the conjugate gradient and GMRES Krylov solvers used are part of the Belos linear solver package within the Trilinos framework (Heroux et al., 2005). In general, the Belos package produces solutions to the neutron diffusion problem nearly two orders of magnitude faster than the MCLS library for serial computations using the conjugate gradient solver. Therefore, when we directly compare these two codes, the MCLS parallel efficiencies will appear artificially inflated relative to the Belos efficiencies due to the fact that the MCLS ratio of work to communication will be much larger than that for the Belos calculations. For every scaling study presented, the calculations were performed 3 times with the average values reported for wall time and those average values used for the efficiency computations. In addition, it should be noted that in practice, symmetric positive-definite problems like the neutron diffusion problem solved here would never be solved using GMRES. Instead, a method such as conjugate gradient would typically be used due to the enhanced performance. We use GMRES here simply as a means of providing an additional comparison for MCSA performance.

5.8.1 Titan

All parallel scaling studies presented in this work were performed on the Titan Cray XK7 machine at the Oak Ridge Leadership Computing Facility located at Oak Ridge National Laboratory. Titan is a heterogeneous architecture machine combining both

[1]MCLS is an open source C++ library built on the Trilinos framework and can be downloaded at `https://github.com/sslattery/MCLS`

CPU and GPU components to achieve a theoretical peak performance of more than 20 petaflops. The technical specifications for Titan for only the CPU portion of the machine are given in Table 5.4.

Processor	16 core AMD
Nodes	18,688 AMD Opterons
Cores/Node	16
Total Cores	299,008
Memory/Node	32 GB
Memory/Core	2 GB
Interconnect	Gemini

Table 5.4: **Titan Cray XK7 hardware specifications.** *Specifications presented for Titan in this table were gathered from the Oak Ridge Leadership Computing Facility at* `http://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/`

For the scaling studies presented in this work, only the CPU components of Titan specified in Table 5.4 were utilized. This is due to the fact that the implementations of the parallel MCSA algorithm in the MCLS library utilized code written only with the Message Passing Interface (MPI) and therefore could not take advantage of the GPU accelerators. Even considering just the CPU portion of Titan, nearly 300,000 CPUs are available to test the parallel performance of the MCSA algorithm at high levels of concurrency.

For each parallel performance metric presented in the previous section, a reference case is required to compute efficiencies for all other cases. For smaller machines, often a serial computation can be performed and used as the base case to get a true measure of speed-up for a particular problem. For our scaling studies, we will always use as a reference case a calculation which uses 16 cores, filling an entire node of the machine. We fill an entire node so that we may ignore the effects on scalability measures that arise from using only a subset of cores in a node. Typically this results in a large drop in efficiency from a serial computation to a calculation that fills the entire node. Once a node is filled, adding subsequent nodes has less of an effect on the scaling, providing a more stable region of analysis.

5.8.2 Communication Parameters

In our description of Brunner and Brantley's algorithm as applied to the Neumann-Ulam method, we noted two free parameters that are user-defined at runtime. The

first is the static data buffer size for communicating histories between neighboring domains and the second is the frequency by which a domain checks for incoming information from neighboring domain, both of which may potentially affect latency and load balancing. Changing the static history buffer size parameter affects the scalability of the algorithm in two ways. If the buffer size is increased, the buffer takes longer to fill and is therefore sent less frequently, potentially causing load balancing bottlenecks where a neighboring domain is idle and waiting for more work to do but is waiting for another domain to complete work. In addition, if the buffer size is too large, the network may become saturated with these very large data packets, slowing down performance. If the buffer size is too small, too many history messages are sent, not providing enough data for a neighboring process to work on to make up for the time spent communicating. Considering the second parameter, when the message check frequency period is increased, more histories are processed locally before looking to do work elsewhere. If this value becomes too large, incoming data buffers may not be cleared frequently enough and cause decreased performance in the domain-to-domain communications. If this value is too small, too much time is spent looking for work instead of actually performing it.

To assess how these parameters affect the performance of the parallel Monte Carlo algorithm on the Titan machine, a parameter study was performed using the neutron diffusion problem on 64 cores (4 nodes). This problem had a reasonably well conditioned spectral radius of 0.78 with a global problem size of 1.6×10^7 DOFs with one stochastic history used per DOF in the MCSA Neumann-Ulam used to compute the correction for the acceleration. Each parameter was varied between 16 and 4096 and the time to solve the problem recorded. The results of this study are presented in Figure 5.15. Compared to the results of Brunner and Brantley for the same type of study, we note qualitatively similar results with the general observation that a larger history buffer check period is better than a shorter one. The larger the check period, the more on-process work is done before checking messages. Second, there was not a significant variation observed in the history buffer size parameter for a given check frequency.

Compared to the reference work, there was also not a significant change in the runtime over the values of parameters tested with all runtimes reported within 6.5% of the fastest case observed. For this work, a history buffer size of 4096 and a buffer check period of 128 performed best for the initial calculations. However, most values in the range tested will give good performance. Brunner and Brantley found that a value of 1024 was best for both parameters. For the results observed here, these

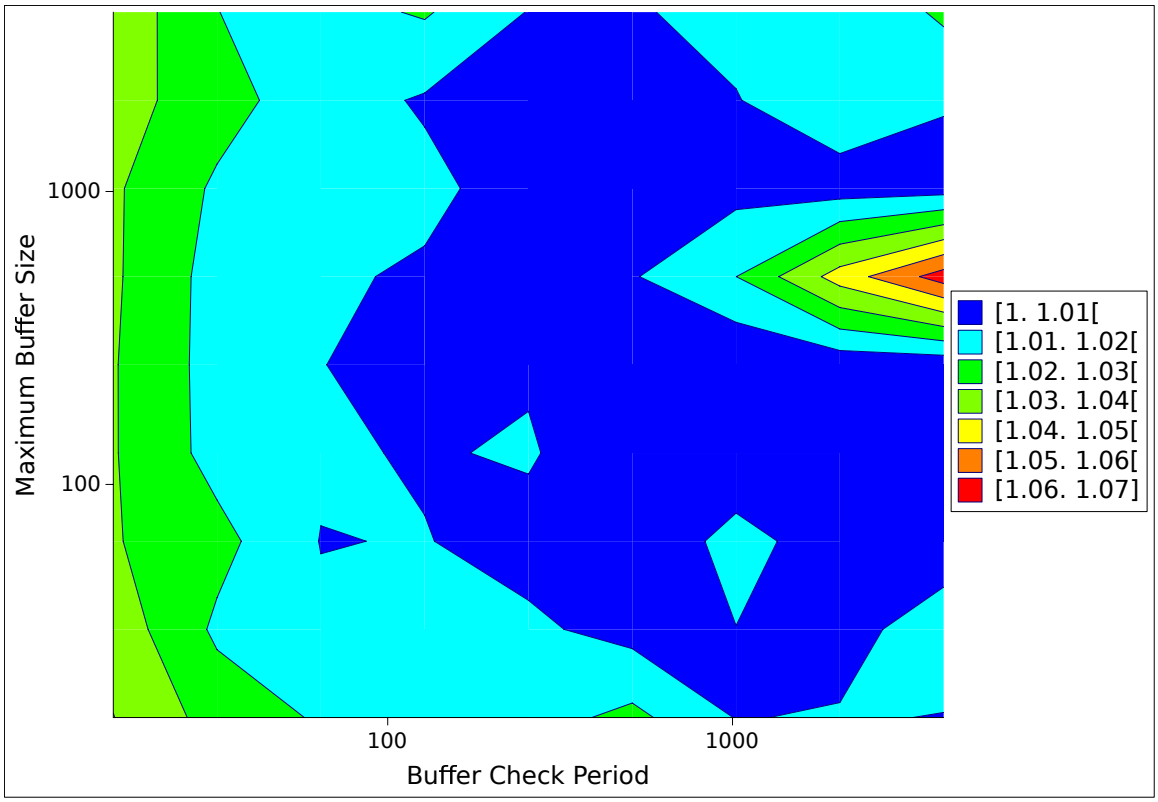

Figure 5.15: **Neumann Ulam parallel algorithm history buffer check period vs. history buffer size sensitivity.** *The color scale represents the ratio of the runtime of a particular parameter variation with the runtime from the faster parameter variation. A value of 1.02 on the color scale corresponds to a runtime 2% longer than the faster time. All runtimes were within 6.5% of the fastest observed case.*

parameters were only 0.7% slower than the fastest case. It may be best practice to fix these parameters internally at 1024 as both this work and the reference found these to perform well several years apart on both different parallel machines and for different transport sequences. Therefore, for the scaling studies presented in the following sections, values of 1024 for both the history buffer size and message check frequency will be used.

5.8.3 Pure Domain Decomposition Scaling

For the first set of scaling studies, we consider the parallel MCSA algorithm where MSOD has not been used in the parallel Neumann-Ulam algorithm. We refer to this case as pure domain decomposition where no overlap or replication has been leveraged. Such a test will isolate specifically the performance of Brunner and Brantley's algorithm as applied to the Neumann-Ulam algorithm. For the strong scaling study, the global diffusion problem size was fixed at 1.6×10^7 DOFs for all solvers and the 16 core run used as the base case. MCSA was used again with one stochastic history used per DOF in the Neumann-Ulam solve to compute the correction for the acceleration with the remainder of the problems and solver parameters the same as those for the verification calculation given in Table 5.1 and Table 5.2. As with all of the scaling studies, 3 calculations for all data points were performed with the average wall time reported and used for the efficiency calculations. Figure 5.16 gives the wall time per iteration for these calculations while Figure 5.17 gives the results of the this study with the given absolute efficiencies computed using Eq (5.23). We report wall time per iteration due to the fact that MCSA converged on the solution in 22 iterations while the Krylov solvers converged in 19 iterations.

There are several key features of Figure 5.16 and Figure 5.17 worth noting. First, up to around 5,000 cores, all methods experience a super-linear improvement in parallel efficiencies. For all methods, this increase is not a function of the algorithmic efficiency as those computed using Eq (5.25) remained at unity, meaning that all computations converged in the same number of iterations relative to the base case[2]. Instead, what is happening here and was observed by Brunner and Brantley in their work is that the large problem used to scale to O(10,000) cores was large enough that memory thrashing was occurring in the 16 core base case. As the number of cores in the strong scaling study was increased and the local problem size decreased, more of the local problem fit into the cache of the system, thus causing improved serial performance and therefore enhanced runtimes. In this region of super-linear efficiencies, we also observe larger efficiencies for GMRES as compared to conjugate gradient. This occurs due to the fact that conjugate gradient has a static set of operations performed at every iteration while GMRES operates on a continually growing subspace. In addition, the orthogonalization operations required to generate that subspace consume much more time than the basic conjugate gradient operations, causing a larger serial runtime and subsequently higher efficiencies. For the same reasons, we also observe

[2]When the algorithmic efficiency remains at unity for all calculations, the implementation efficiency is equivalent to the absolute efficiency and therefore is not presented here.

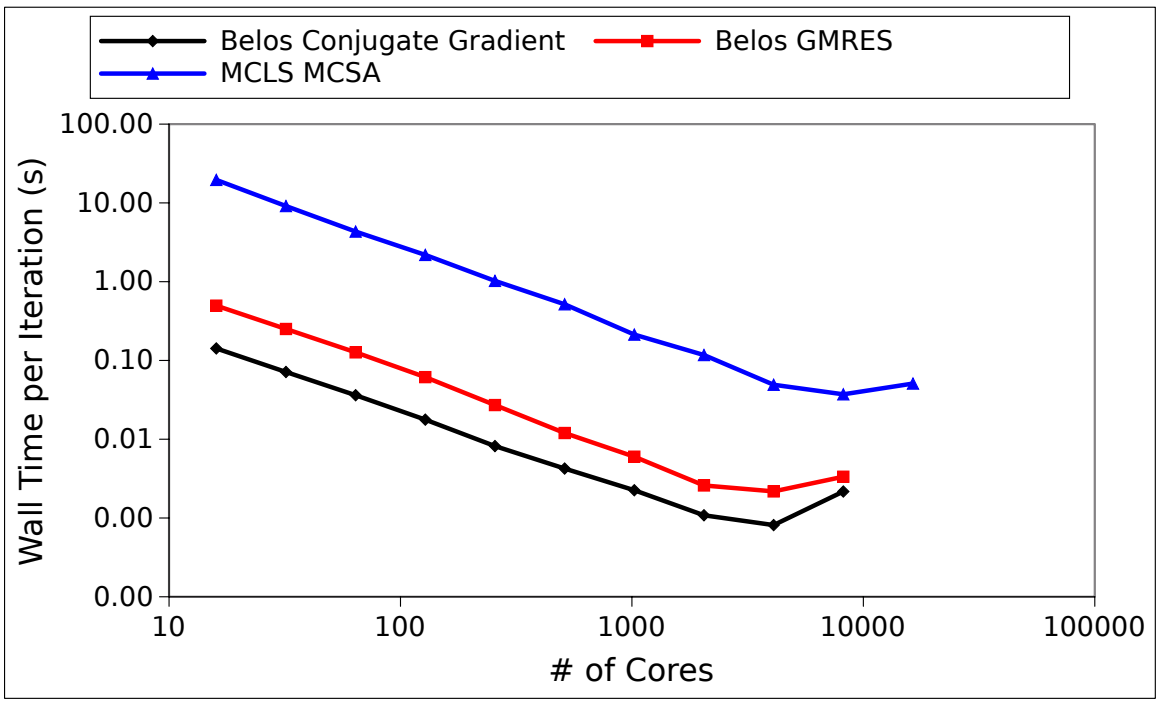

Figure 5.16: **Pure domain decomposition strong scaling wall time per iteration.** *Wall time is reported per iteration because the methods converged in different numbers of iterations. MCLS is over an order of magnitude slower arithmetically than the Krylov solvers. GMRES executes more operations than conjugate gradient and is therefore slower as well.*

higher efficiencies for the MCSA implementation due to the serial performance of the implementation.

Beyond around 5,000 cores, the global problem size selected for this strong scaling study causes the local problem size to shrink enough that parallel communication costs begin to overtake arithmetic costs on process. This takeover causes this effective strong scaling wall that is observed in all methods. Using any more cores to solve this particular global problem is not an efficient use of those parallel resources. Furthermore, the fact that we see a very similar trends in efficiencies for all methods means that they are bound by the same parallel performance limitations in a strong scaling environment. This is important as we can expect the same qualitative scaling performance from MCSA if arithmetic optimization has been performed to improve the serial performance of the implementation. In addition, these results are valuable in that they show MCSA, when properly parallelized, has the ability to be competitive on leadership-class computing platforms with conventional subspace methods with good strong scaling observed in the pure domain decomposition case to O(10,000)

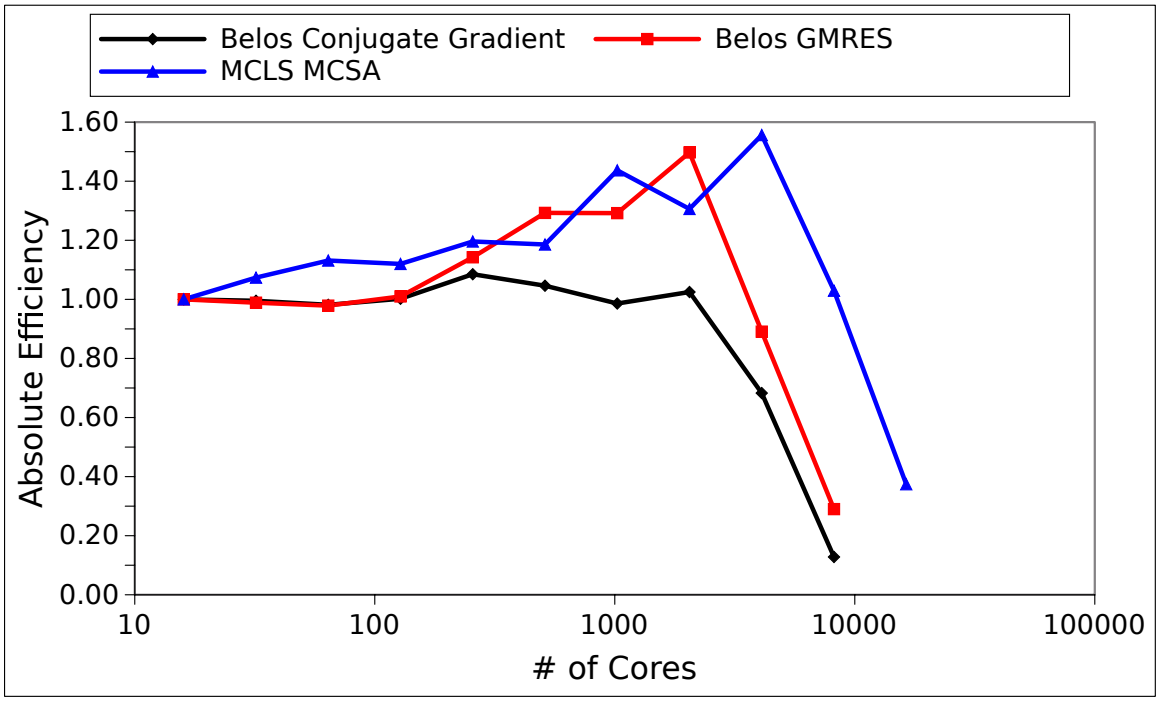

Figure 5.17: **Pure domain decomposition strong scaling absolute efficiency.** *Super-linear speed-up is from memory thrashing in the base case. MCLS is over an order of magnitude slower arithmetically, causing the higher efficiency. GMRES executes more operations than conjugate gradient, also creating a higher efficiency.*

cores.

Next, we consider the weak scaling performance of the algorithm. In this case, we fix the local problem size at 4×10^4 DOFs and increase the number of processors used in the simulation. Figure 5.18 gives the wall time per iteration and Figure 5.19 gives the weak scaling absolute efficiencies for this problem as computed by Eq (5.27). In general, the relative trends when comparing MCSA to the Krylov methods are the same with MCSA and GMRES scaling better than the conjugate gradient method due to the increased amount of arithmetic work on-process. In addition, it is important to note that the weak scaling of MCSA when properly parallelized remains competitive with the Krylov methods keeping in mind the artificial inflation of efficiencies.

As an additional means of comparison between MCSA and the Krylov methods, we compute the algorithmic efficiencies of the calculations using Eq (5.25). As shown in Figure 5.20, the algorithm efficiency of the Krylov algorithm improves as a function of the number of cores in the problem due to the fact that the number of iterations required to converge decreased from 20 in the base case to 19 and then to 18 for the

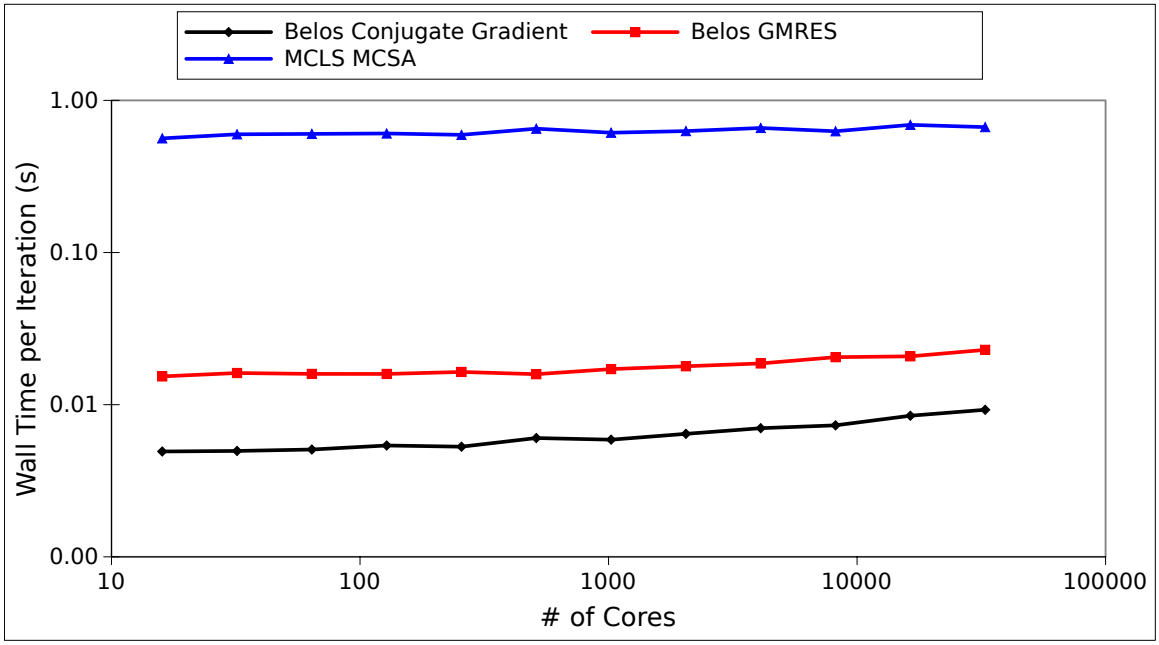

Figure 5.18: **Pure domain decomposition weak scaling wall time per iteration.** *Wall time is reported per iteration because the methods converged in different numbers of iterations. MCLS is over an order of magnitude slower arithmetically than the Krylov solvers. GMRES executes more operations than conjugate gradient and is therefore slower as well.*

final set of weak scaling computations. When this reduction in iterations is used to account for the observed runtimes and computed absolute efficiencies for weak scaling we end up with the implementation efficiencies computed using Eq (5.26) given in Figure 5.21. Because MCSA maintains the same iterative performance as a function of global problem size in the weak scaling study, the implementation efficiencies compare more favorably to the Krylov methods. Using the implementation efficiency, we are effectively measuring the parallel performance of a single iteration for each method by looking at these implementation efficiencies. As the global problem size increases, MCSA is less prone to a drop in parallel efficiency. If optimization has been performed and runtimes potentially approach those of the Krylov methods, this comparison may not be as favorable. Overall, weak scaling performance of the parallel MCSA algorithm is excellent for the pure domain decomposition case with absolute parallel efficiencies of over 80% observed at over 32,000 cores.

In addition to comparing MCSA scaling to conventional methods, we also motivate avoiding any type of master/slave algorithms at these high levels of concurrency by comparing the binary communication tree implementation used in the previous results

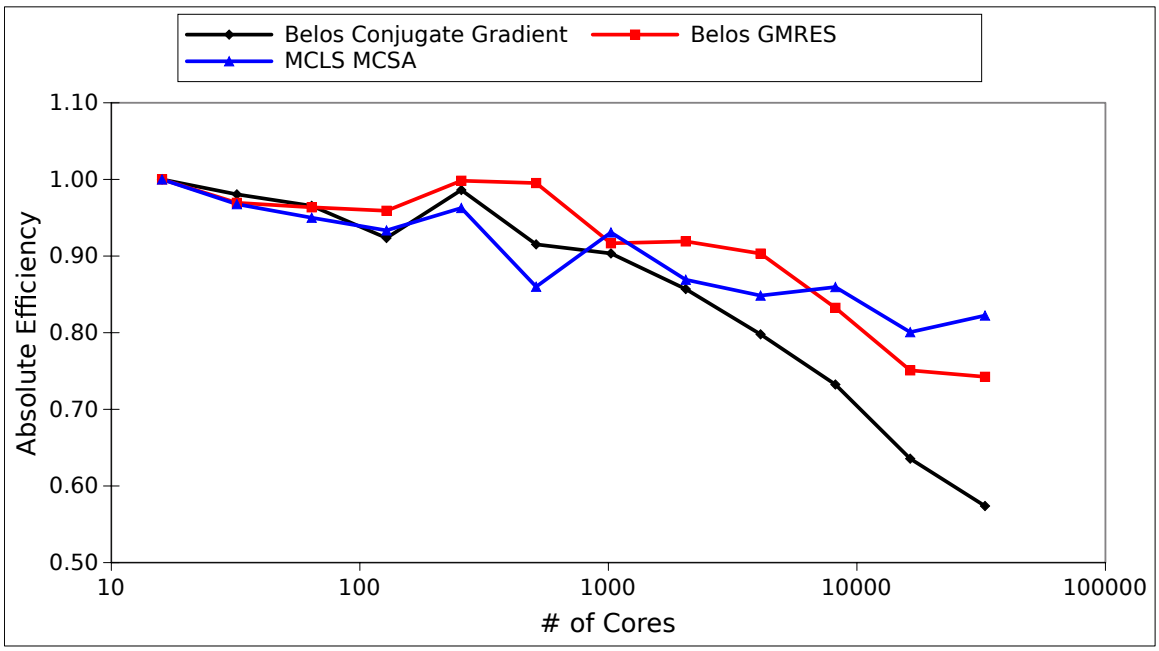

Figure 5.19: **Pure domain decomposition weak scaling absolute efficiency.** *MCLS is an over order of magnitude slower arithmetically, causing the higher efficiency. GMRES executes more operations than conjugate gradient, also creating a higher efficiency.*

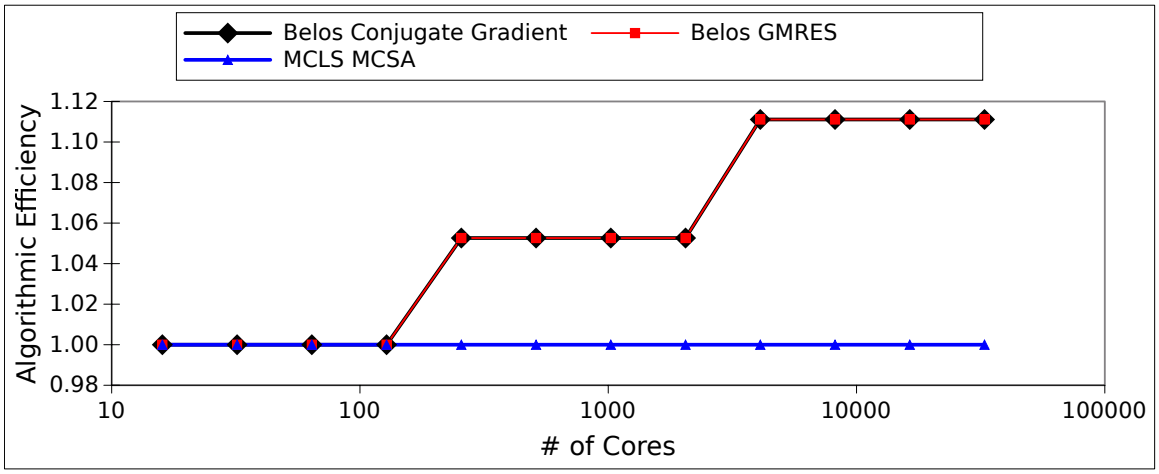

Figure 5.20: **Pure domain decomposition weak scaling algorithmic efficiency.** *As the global problem size was increased in the weak scaling study, both conjugate gradient and GMRES converged in fewer iterations, causing the rise in algorithmic efficiency.*

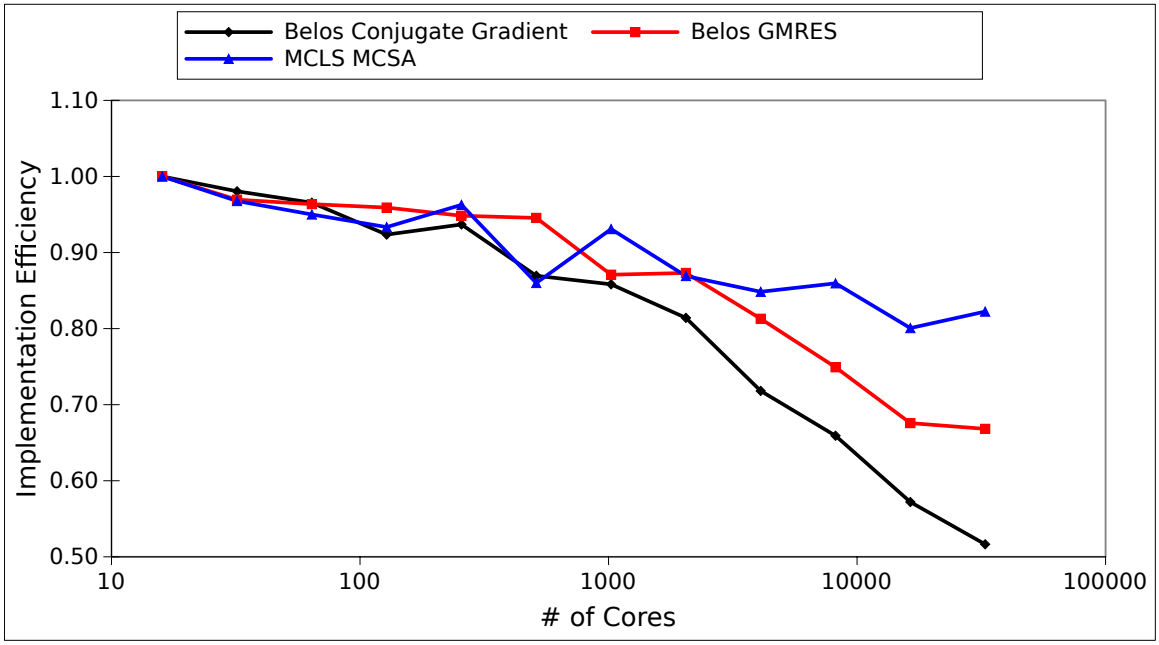

Figure 5.21: **Pure domain decomposition weak scaling implementation efficiency.** *Considering the scalability of a single iteration of the implementation, the inflated efficiencies of the MCLS MCSA implementation become more obvious. Implementation optimization will likely reduce the MCSA values to a level more comparable to the Belos values.*

with a parallel MCSA implementation that uses the master/slave scheme to determine the end of a history transport stage as shown in Figure 5.3.2 instead. Figure 5.22 provides this comparison using the strong scaling study while Figure 5.23 gives the results of the comparison for the weak scaling study. In both cases, the deficiencies caused by using the master/slave scheme for transport completion are obvious. At around 1,000 cores, both the strong and weak scaling performance seriously degrades while the binary tree pattern maintains excellent performance. At this number of cores, any extra work given to the master process, even if it amounts to summing a handful of integers and checking for a few extra messages as in this algorithm, grows enough to create serious load imbalance and reduce performance.

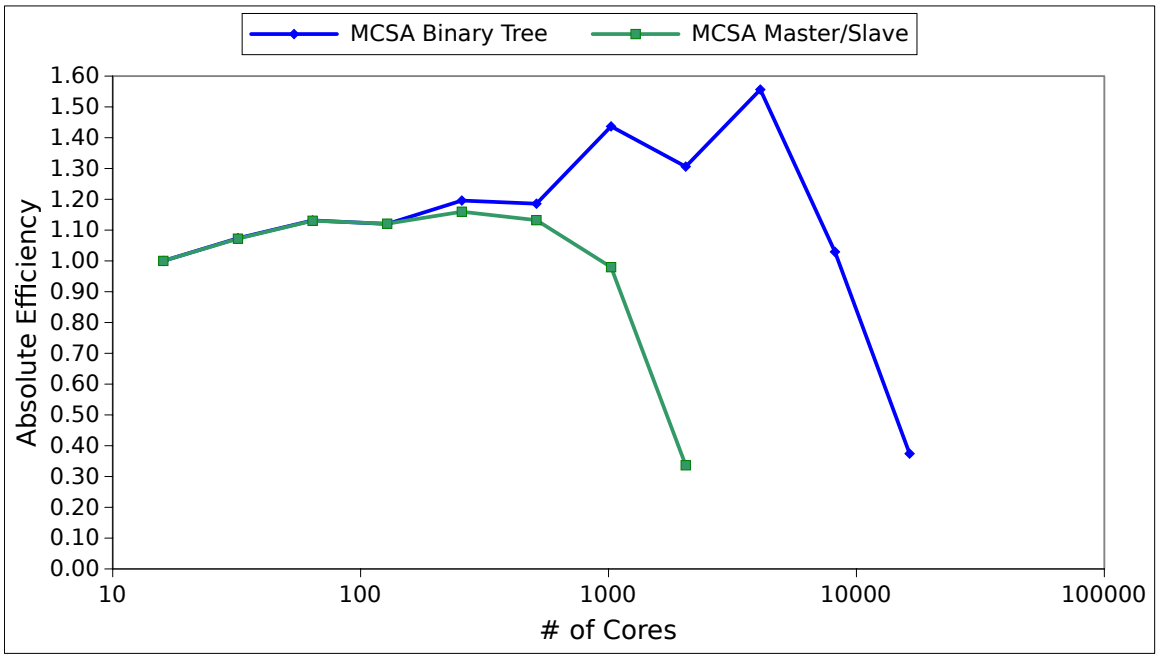

Figure 5.22: **Binary tree vs. master/slave communication scheme strong scaling absolute efficiency.** *The master/slave scheme will not scale beyond* $O(1,000)$ *cores due to load imbalance.*

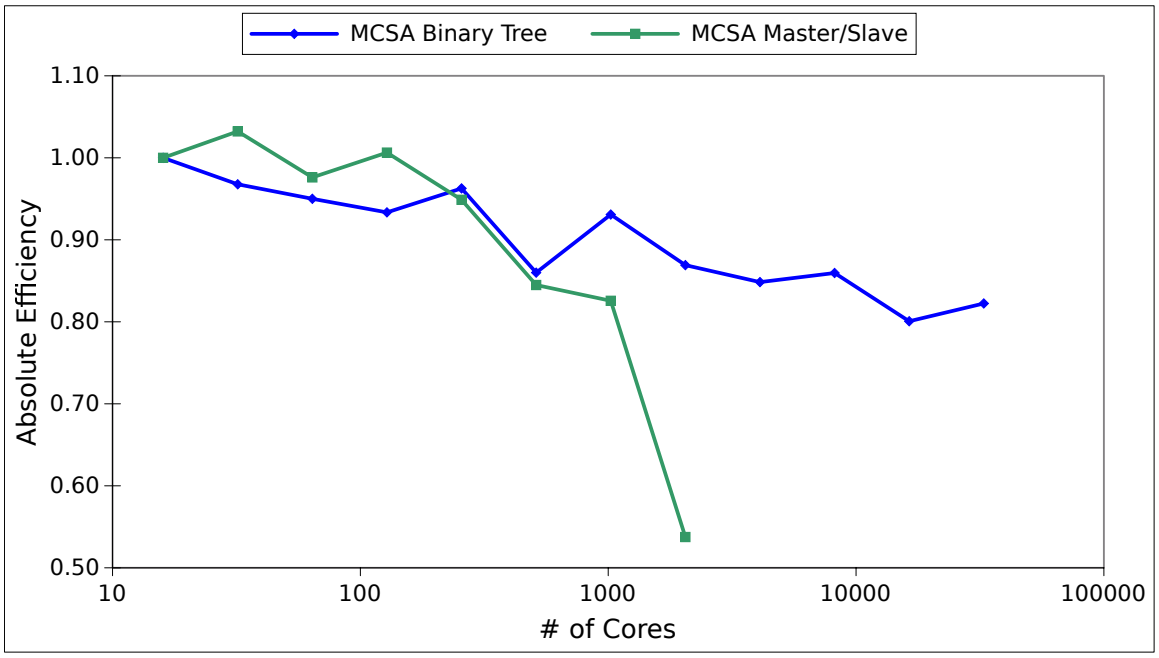

Figure 5.23: **Binary tree vs. master/slave communication scheme weak scaling absolute efficiency.** *The master/slave scheme will not scale beyond* $O(1,000)$ *cores due to load imbalance.*

5.8.4 Overlapping Domain Decomposition

In the next set of scaling studies, we will explore how adding overlap to the subdomains in the problem affects runtimes and parallel performance as compared to the case with pure domain decomposition. To do this, we utilize the set of analytic relations developed in § 5.2 to design these numerical experiments. For the neutron diffusion problem used in these scaling studies, the linear operator had a spectral radius of 0.787 and a weight cutoff of 1×10^{-2} was used with the Neumann-Ulam solver. Using Eq (5.12) gives an expected random walk length of 19.22 transitions for any given history in the problem. Using the mean-chord approximation given by Eq (5.21), we then expect approximately 5.3% of the histories born in a particular domain to leak out after the first transport stage. By adding overlap, we aim to reduce this fraction of communicated histories and therefore boost the parallel performance of the algorithm.

To determine the amount of overlap to test for the scaling studies, we use a variation of Eq (5.15) to determine how many discrete states a history will move on average from its birth site. We modify Eq (5.15) to only account for discrete movements:

$$\sqrt{\bar{n}_k^2} = n_s\sqrt{k}\,, \tag{5.28}$$

where $\sqrt{\bar{n}_k^2}$ is now the root-mean-squared number of discrete states a history will move from its birth site. For our diffusion problem used in these scaling studies, we compute a value of 2.63 states per history for any of the dimensions of the system. This means that we can expect to end up on average 2.63 states in both the i and j dimensions away from the starting state. Therefore, we will use the overlap generation algorithm outlined in § 5.4.1 such that we parametrically grow the local domain using 1, 2, 3, and 5 levels to measure how scalability is affected when this root-mean-squared number of states is considered. For example, using 3 levels of overlap should keep over 50% of the histories that would be communicated in the pure domain decomposition case on-process as this is greater than the root-mean-squared number of states we would expect a history to travel.

For the performance metrics, we may use the same weak and strong scaling efficiency metrics that we used for the pure domain decomposition case. As we are comparing to the case without overlap and are looking to boost performance relative to the case without overlap, for both weak and strong scaling calculations we will use the 16-core case without overlap as the reference computation for all efficiency measures reported.

The absolute efficiency results from the strong scaling exercise are given in Fig-

ure 5.24. Aside from the boosting of efficiencies at 2,048 cores, little evidence of strong scaling improvement is evident for all levels of overlap tested. To further explore any potential benefits of adding overlap into the system, Figure 5.25 gives the difference in the efficiencies between all overlap cases and the reference case without overlap at each core count. At smaller numbers of cores, there is no observed benefit to using overlap to improve strong scaling. The improvements noted at 2,048 cores are also present here and account for the best efficiency improvements for all core counts tested. at 16,384 cores, using 2 levels of overlap did give a 6.4% boost in efficiency while at 1,024 cores this same amount of overlap actually reduced the efficiency. Beyond 2 levels of overlap, efficiency levels begin to fall with 5 levels of overlap performing the worst for all cases. From this, we don't expect adding any more overlap will improve the scalability of the algorithm and the best strong scaling performance was observed when the number of overlap levels is approximately equal to the root-mean-squared number of states we expect a history to move from its starting point as computed by Eq (5.28). We will defer an explanation of the drop in efficiencies for levels of overlap beyond this value until we have analyzed the weak scaling data.

For weak scaling, the same study as in the pure domain decomposition case was performed again with 1, 2, 3, and 5 levels of overlap. The absolute parallel efficiencies from these calculations are given in Figure 5.26 and the differences in efficiencies from the calculations without overlap. The results here are much the same in that there is not a dramatic improvement in scalability relative to the case without overlap. Opposite the strong scaling calculation, overlap enhances weak scaling efficiencies at lower core counts and degrades the efficiencies at higher core counts. In addition, 1 level of overlap provided the best enhancement of scalability when overlap was beneficial.

For both the strong and weak scaling studies, the parallel performance of the algorithm was either marginally improved or not improved at all. We would always expect an improvement in performance for using overlap to prevent the transport of histories from domain to domain during execution of the Neumann-Ulam algorithm. The reduction in transport means a reduction in parallel communication and therefore should never degrade the performance of the algorithm. It is true that doing so in fact enhances Monte Carlo performance. However, the Neumann-Ulam algorithm is not representative of the entire MCSA algorithm. When overlap is used, an overlapping tally vector is constructed in parallel such that multiple cores may own multiple states in the system. When this occurs, the tally data associated with these shared states must be communicated after the Neumann-Ulam solve as outlined in § 5.4.2. This

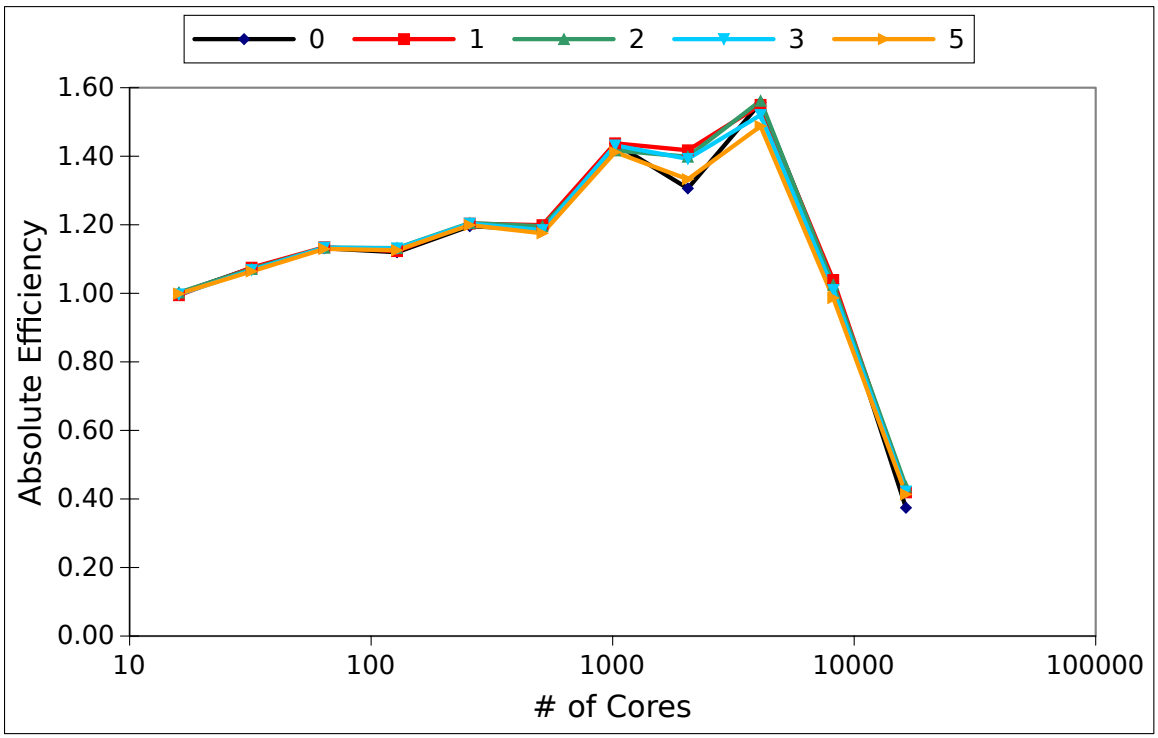

Figure 5.24: **Strong scaling absolute efficiency with varying levels of overlap.** *Values of 0, 1, 2, 3, and 5 levels of overlap were used as the problem had a root-mean-squared number of states of 2.63 for each history.*

reduction operation then builds the full MCSA correction vector in the decomposition of the solution vector.

Therefore, by building overlap we are in fact reducing parallel communication during the Neumann-Ulam solve but we are simply deferring that communication until after Monte Carlo transport when the overlapping tally vector must be reduced. When overlap becomes large enough (i.e. larger than the root-mean-squared number of discrete states a history will travel), then the savings in communication during the Neumann-Ulam solve start to become smaller than the communication created in the overlapping tally vector reduction. We see this in both the strong and weak scaling cases at larger levels of overlap. For the strong scaling case, we see better performance improvement at higher core counts because the total number of overlapping states on each core is smaller from the local problem size shrinking. For weak scaling, we see better performance improvement at lower numbers of cores because although the local overlap size is fixed, the global amount of overlap is growing with core count and therefore reducing the performance of the tally reduction operation.

Based on this data, a few levels of overlap within the expected distance of travel

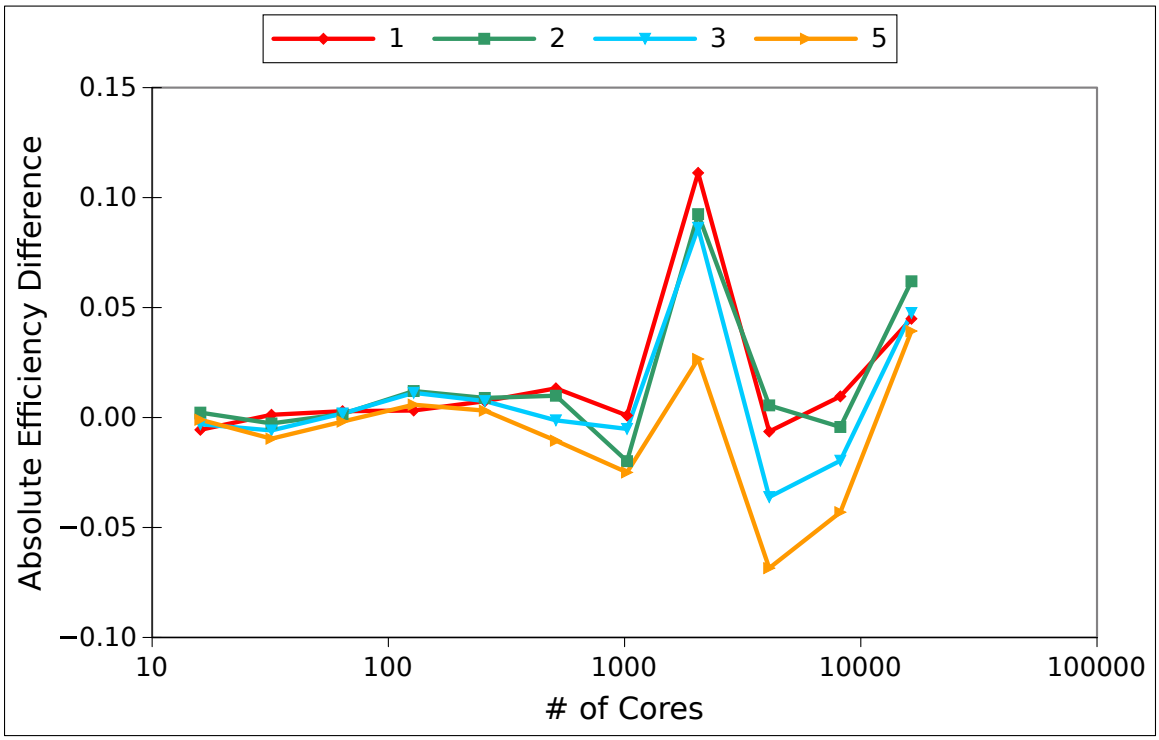

Figure 5.25: **Strong scaling absolute efficiency difference with respect to the 0 overlap base case.** *Values of 1, 2, 3, and 5 levels of overlap are compared here with the absolute difference of the absolute efficiencies presented.*

for a history may offer some marginal improvement in scalability, however, it is not as effective as one might think given its improvement to reactor physics calculations as observed in the literature. This is largely due to the density of states in the system that creates the described performance bottlenecks in the overlapping tally vector reduction operation. In a Monte Carlo particle simulation, these overlapping tally reduction operations are not expected to be performed at the speed they are required in MCSA. For example, in a criticality calculation with overlapping domains, we would expect to perform as many parallel tally vector reduction in a particle transport simulation as there are particle stages required for convergence of the eigenvalue. However, these particle stages will likely be much more costly in terms of wall time relative to a Neumann-Ulam solve in an MCSA iteration due to the more extensive particle physics and logic structure that must be evaluated. Therefore, we can expect this overlapping tally reduction to be less significant to scalability in those operations. We do not have as complicated a logic structure in a history random walk for the Neumann-Ulam method and therefore overlap can inhibit scalability if too large.

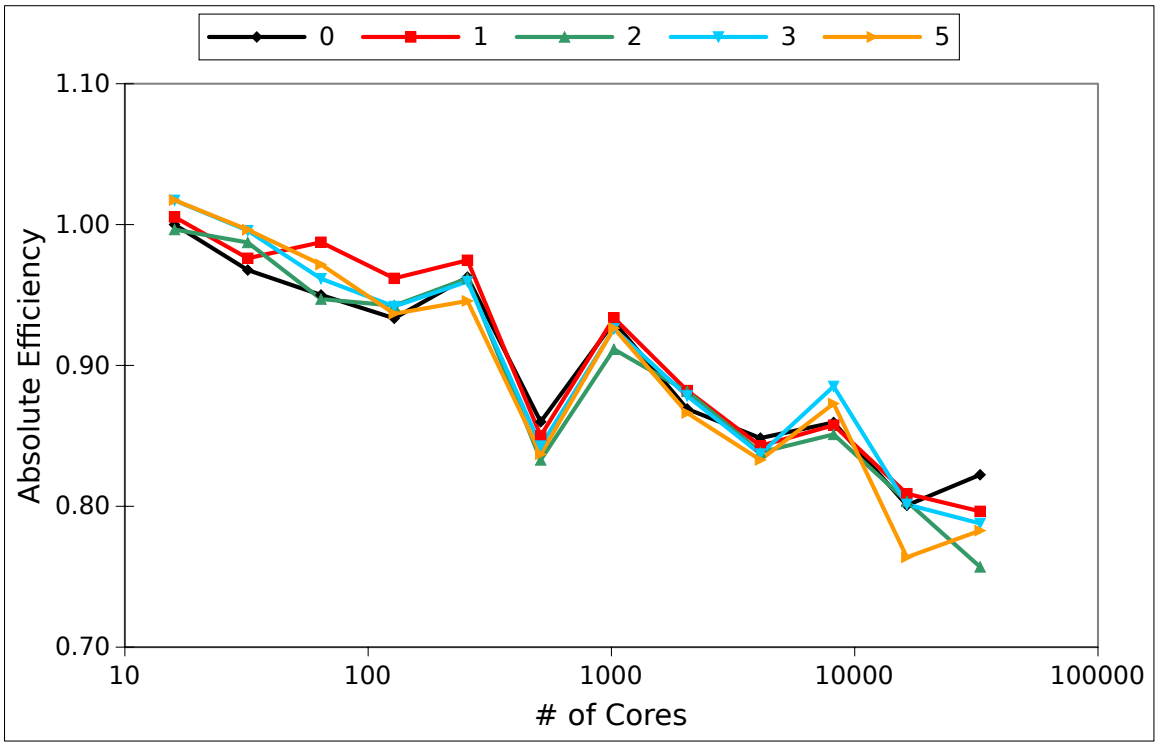

Figure 5.26: **Weak scaling absolute efficiency with varying levels of overlap.** *Values of 0, 1, 2, 3, and 5 levels of overlap were used as the problem had a root-mean-squared number of states of 2.63 for each history.*

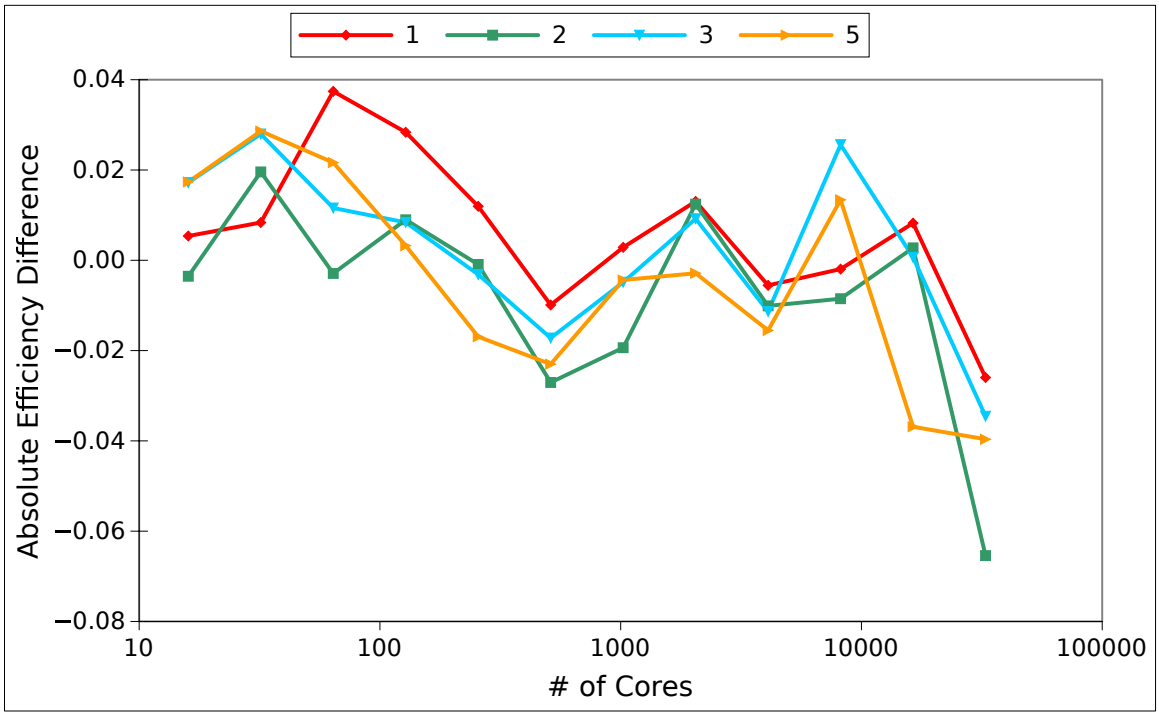

Figure 5.27: **Weak scaling absolute efficiency difference with respect to the 0 overlap base case.** *Values of 1, 2, 3, and 5 levels of overlap are compared here with the absolute difference of the absolute efficiencies presented.*

5.8.5 Multiple Set Domain Decomposition

We next look to apply replication to the problem in the form of multiple sets. As the previous analysis showed overlap to be of little assistance in improving scalability, we only apply multiple sets to the problem with pure domain decomposition within the sets for these scaling studies. Before performing parallel scaling studies using multiple sets, several aspects of the replicated parallel problem must be considered in order to effectively craft the studies and interpret their results. First is the question of how many stochastic histories to use to generate the correction vector in the MCSA iteration. For this work we consider two choices; the split case and the fully replicated case. As an example of these cases, consider a single set problem where 20,000 histories are used in the Monte Carlo solve. If we choose the split case, then solving the same problem with 2 sets would give 10,000 histories in each set and 4 sets would give 5,000 histories in each set. Splitting the histories in this way provides the same global number of histories used to compute the correction in the single set problem. We expect to reduce the time to solution by doing this as each individual set solve will occur faster in an almost linear fashion with the reduced number of histories. For the replicated history case, each additional set in the problem will use just as many histories as the single set case. Using 2 sets with fully replicated histories for our example problem then has each set computing 20,000 histories with 40,000 global histories in the computation. For the 4 set case, the number of global histories is increased to 80,000. We also expect this type of replication to reduce the time to solution as the extra global histories should reduce the stochastic uncertainty in the correction vector and reduce the overall number of MCSA iterations required to converge.

5.8.5.1 Strong Scaling with Multiple Sets

For the strong scaling study, we apply the additional sets as follows. Using the same problems as the pure domain decomposition scaling study, the multiple sets are used to replicate those problems and boost the core count. For example, for the 4 set case, the data point at 64 cores has the same N/P ratio in one set as the single set problem with 16 cores. This is due to the fact that N is fixed for the strong scaling study and 4 replications gives a set size of 16 cores. For the 2 set case, the 32 core data point is solving the same N/P problem in a set as the 16 core base case.

In addition to considering the replicated domain, we must also consider how to modify the efficiencies we will compute. For the strong scaling case, recall that the

objective is to decrease the time to solution for a given global problem size N by increasing P. Therefore, we may use Eqs (5.24), (5.25), and (5.26) to compute the various strong scaling efficiencies, but we must modify the reference computation in each of these formulas to reflect our objective. For all efficiency computations, the 16 core single set computation will be used as the reference case. By adding more cores to the problem, either by replicating with multiple sets, or decreasing the local problem size, any computational resources added to the problem should aim to result in speedup of this reference case. If this speedup for one technique of adding cores is less than another, then that technique is deemed a less efficient use of those computational resources.

For the multiple sets problem, the single set strong scaling study from the pure domain decomposition base case was repeated for multiple sets with both splitting of histories across sets and full replication of histories across sets using both 2 and 4 sets. Figure 5.28 gives the actual wall times for these computations. There are two important features to note in the wall time data. First, splitting the histories among sets instead of replicating them is faster due to the fact that cutting the number of histories in half reduces the time of an iteration by a factor that is nearly linear while increasing linearly the number of histories through replication does not decrease the iterations required to converge by a linear factor. Second, until the strong scaling wall is hit starting at around 10,000 cores, the single set case actually performs the fastest with only marginal improvement in runtimes at higher numbers of cores with more sets. At these larger numbers of cores, however, the improvements are so modest that one is better off running the same problem using a single set at lower core counts to conserve computational resources while maintaining a good time to solution.

From this timing data we then compute the absolute efficiencies for strong scaling as given in Figure 5.29 using the 16-core single set calculation as the reference case. As observed in the timing data, because the single set case had the fastest run times up to the strong scaling wall it also has the best observed efficiencies. For the 2 set case with history splitting, although the time to perform an iteration was nearly cut in half, the extra parallel overhead associated with reducing the tally across the blocks to combine the set results increases run times and reduces efficiencies. This reduction across blocks is even more obvious for the 4 set case. For the replicated cases, we do observe a flat region of parallel efficiencies at larger core counts (up to around 32,000 for the 4 set case with history replication) showing improved scaling within context of not considering the single set calculation. Unfortunately, the additional cores do not reduce the runtime of the global problem as efficiently as expected. Ultimately,

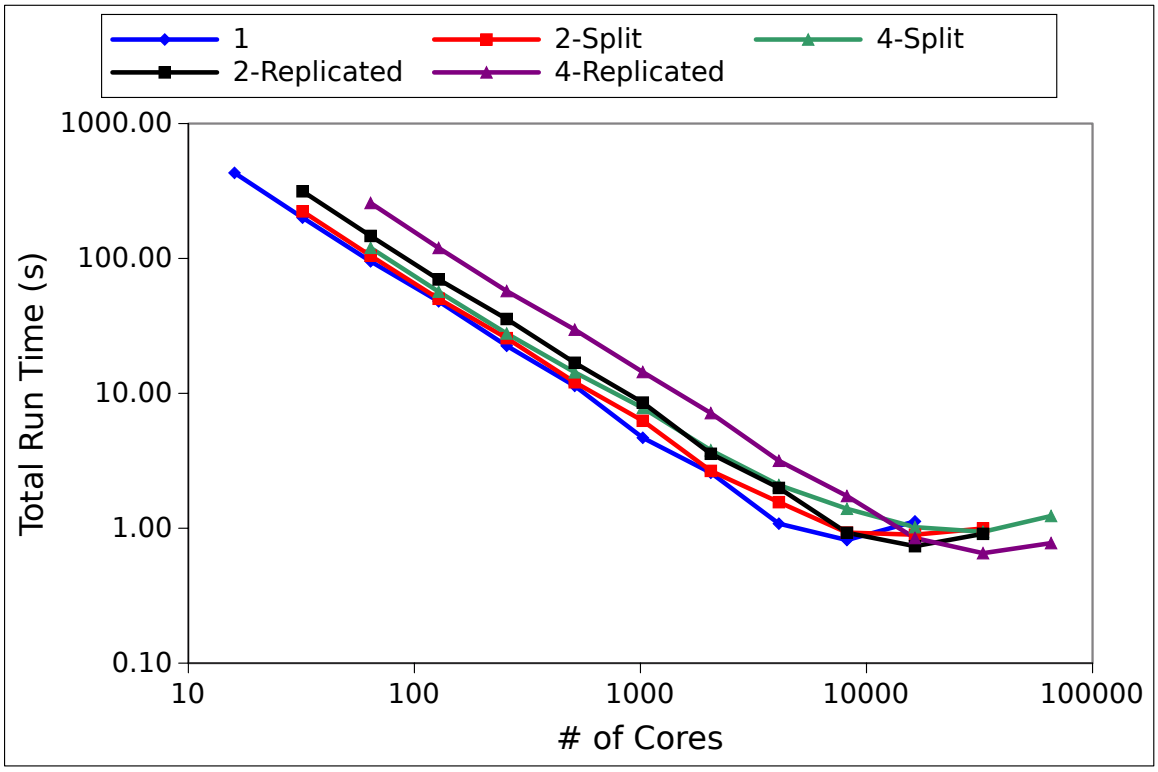

Figure 5.28: **Wall time in seconds to solution for each case for the strong scaling study with multiple sets.** *Until the strong scaling wall is reached at O(10,000) cores, the single set case performs the best. For the multiple set cases, it is faster to split histories among sets and maintain iterative performance rather than replicate histories to improve iterative performance.*

all cases effectively hit the strong scaling wall at the same time at around O(10,000) cores.

We expect this if we consider the fact that adding sets to the problem is in itself effectively a form of strong scaling. For any given problem of fixed size N, adding extra sets to the computation decreases the global ratio of N/P as P will grow with the addition of sets (although N/P is constant within a set). For the split history case, the splitting further exacerbates this situation by further decreasing the amount of histories that will be computed in the local domain, giving a larger ratio of communication to work in the Monte Carlo solve. As with with single set results, we also observe some efficiencies greater than unity for the multiple set cases also due to improved cache utilization.

Although not as dramatic as one might originally expect from introducing replication, there are some modest gains in efficiencies at high core counts by using multiple

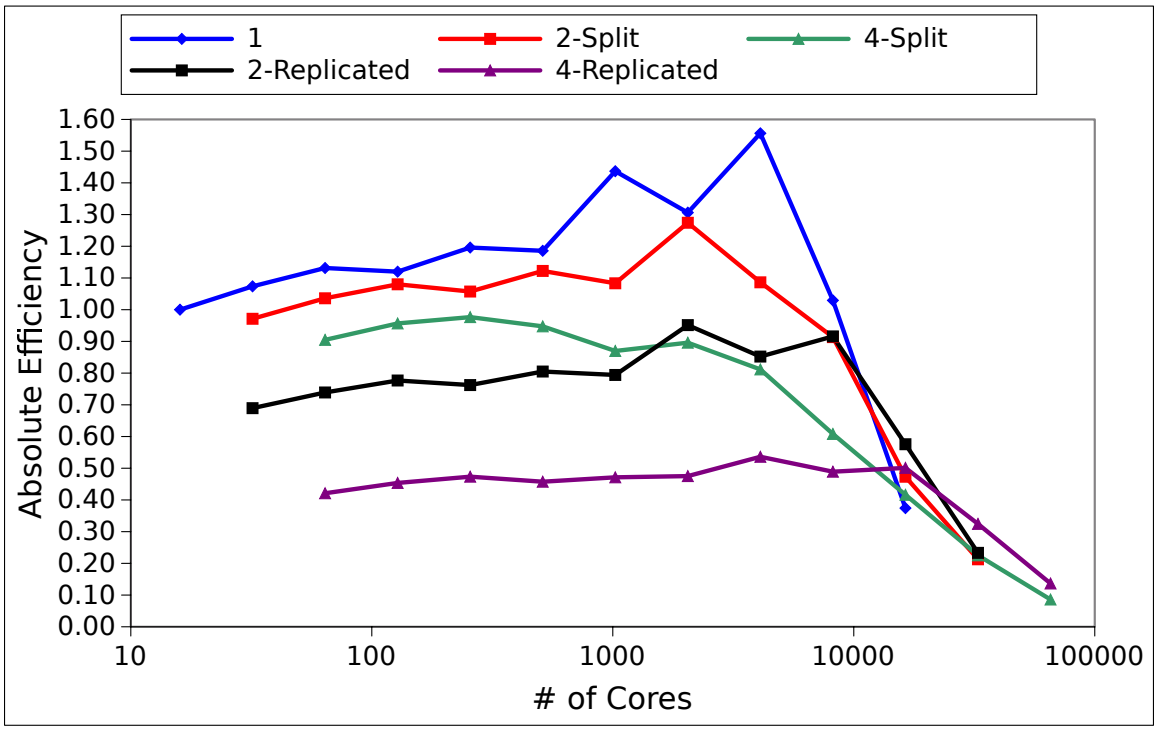

Figure 5.29: **Multiple set absolute parallel efficiency.** *Computed relative to the 16-core 1-set base case. Using a single set is the best means by which to improve strong scaling up to the strong scaling wall. At higher core counts past where the single set case hits the strong scaling wall adding sets can provide some improvements in efficiencies.*

sets. Of interest here is where the multiple set cases perform better than the single set case after the single set case has hit the strong scaling wall. In Figure 5.29, at 16,384 cores the single set case operates at 38% efficiency while at the same number of cores, the 2 set case with history replication operates at 62% efficiency. This is reasonable improvement in the use of those computational resources, gained by replicating the problem. Beyond 16,384 cores, there is not a compelling reason to use more computational resources for the this problem.

Also of interest here is the effect of splitting histories among sets versus replicating them on the iterative performance of the method. Table 5.5 gives the iterations required to converge and computed algorithmic efficiencies for each of the cases presented in the strong scaling data. As expected, splitting histories maintains a fixed global number of histories and therefore maintained the number of iterations required to converge relative to the single set base case. When histories were replicated, adding sets decreased the number of iterations required to converge in a nonlinear fashion.

Therefore, increasing the number of histories per MCSA iteration improves algorithmic efficiency. Using these algorithmic efficiencies, the implementation efficiencies for the multiple set cases were computed and are given in Figure 5.30. From these results, we see that the replicated history cases are even less efficient than in the absolute case with respect to the performance of a single iteration. Not only do we have the additional parallel overhead of doing the block level reduction, but we also have additional histories that increase the time to solution for a single iteration.

Case	Iterations	η_{alg}
1	22	1.0
2-split	22	1.0
2-replicated	16	1.375
4-split	22	1.0
4-replicated	13	1.692

Table 5.5: **Strong scaling algorithmic efficiency using multiple sets.** *Replicating histories improves algorithmic efficiency by converging in fewer iterations.*

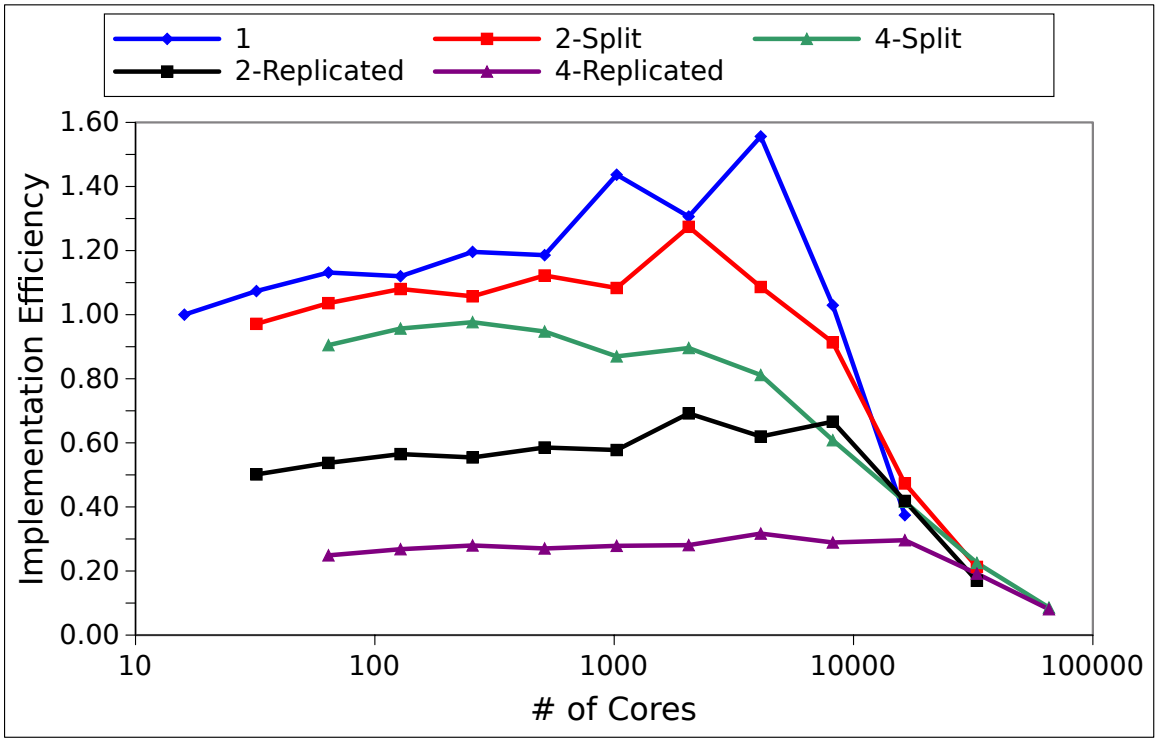

Figure 5.30: **Multiple set strong scaling implementation efficiency.** *Computed relative to the 16-core 1-set base case. Considering iterations required to converge, the performance of the split history cases at much better than the cases with history replication.*

5.8.5.2 Weak Scaling with Multiple Sets

Next, we continue to a weak scaling study of the multiple sets problem. In the same way as the strong scaling study, we will modify the pure domain decomposition study to account for the additional replication. For each calculation in the single set study, that problem will be replicated either 2 or 4 times with both split and replicated histories to assess the effects on scalability and runtime. Figure 5.31 gives the wall times from the multiple sets weak scaling study. Of primary importance here is the fact that adding sets to the problem with both split and replicated histories actually decreases the time to solution. This was not observed in the strong scaling case until much larger core counts as the multiple set computations hit the strong scaling wall after the single set computation.

This result is significant in that it shows a solution technique in which the linear problem can actually be replicated and enhance time to solution, something that cannot be achieved with conventional methods. At larger core counts, however, this reduction in compute time is not as large as at lower core counts due to the fact that the tally reduction across blocks requires more resources and those resources are clearly growing with core count (most obviously in the 4 set case with split histories). In addition, at 131,072 cores for the largest 4 set computation, load balance is likely becoming an issue due to the linear system and subsequent parallel matrix and vector operations only occurring on a subset of the entire processor space of the problem while the Monte Carlo calculation is occurring on all processors.

For a multiple set analysis of the weak scaling timing results, we again have to consider that adding sets to the problem is actually a form of strong scaling. Consider a single set problem ($S = 1$) for a given P and N. When that problem is now solved using 2 sets we have $S = 2$, N and $2P$. For any number of sets we then have a global problem size of N while the number of processors is SP where P is the number of processors in the single set case. Therefore, N/P is no longer fixed in the weak scaling study but rather $N/(SP)$ is fixed. As a result, we modify the weak scaling absolute efficiency relationship to account for this fact:

$$\eta_{weak}(M|N,P|Q,S) = \frac{T(M,Q)}{S \times T(N,P)} \,, \tag{5.29}$$

where now the absolute efficiency is a function of the number of sets in the problem and again $M/Q = N/P$. As with the strong scaling study, we will use the 16-core single set case as the reference computation for efficiencies. Interestingly, Eq (5.29) has a very similar form to the strong scaling efficiency given by Eq (5.23), representative

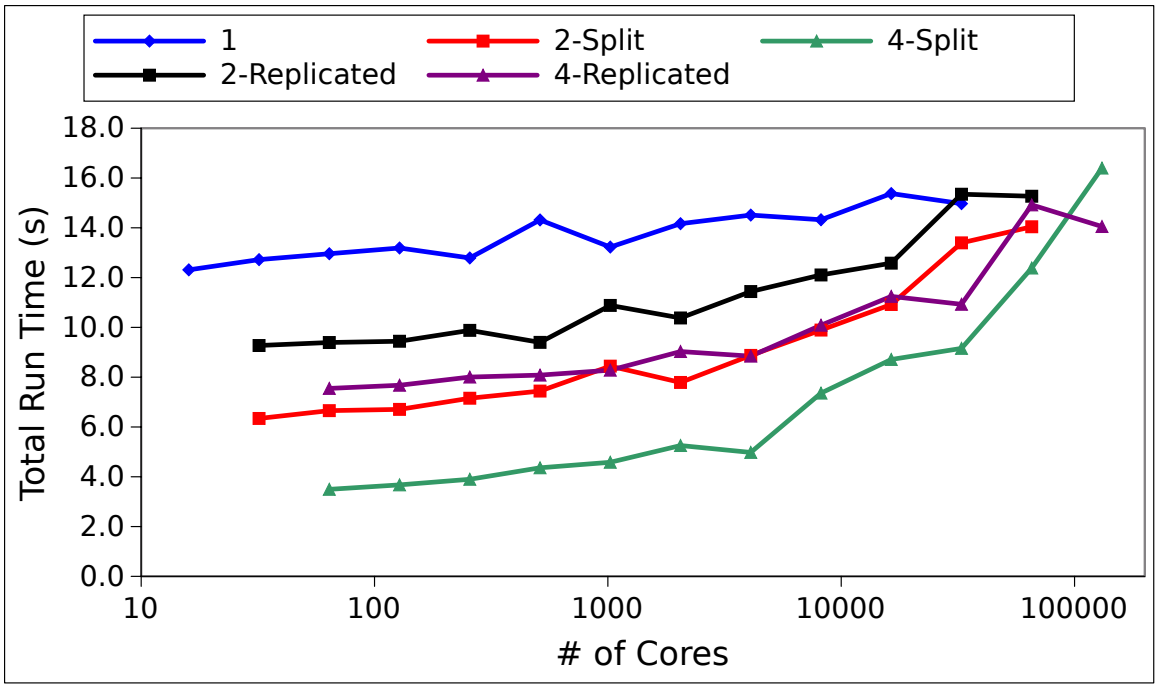

Figure 5.31: **Wall time in seconds to solution for each case for the weak scaling study with multiple sets.**

of the strong scaling nature of adding additional sets. For problems where adding sets improves time to solution, this new weak scaling formulation accounts for how efficiently those extra resources are incorporated into the problem. Without such a modification, the largest case run for the split 4 set case would have a computed weak scaling efficiency of nearly 100% using the standard formula for absolute efficiency. However, using 4 times as many cores as the single set case increased the runtime for a fixed global problem size and therefore we would actually expect to compute a very low efficiency that shows this very poor use of resources.

Figure 5.32 gives the results of the weak scaling efficiency computations using the multiple set metric given by Eq (5.29). Although we do reduce the time to solution over the base cases for nearly all data points, the observation here is that the reduction in time to solution is still not large enough to efficiently use all resources. For example, for the 2 set case to be a perfect use of resources with either split or replicated histories, the runtime of that calculation must be half of that for the single set case. For the 4 set case, this run time should be a quarter of the single set case for perfect scaling. This was not the observation for the runtimes in Figure 5.31 and therefore the single set case is still the most efficient use of computational resources in this scaling study.

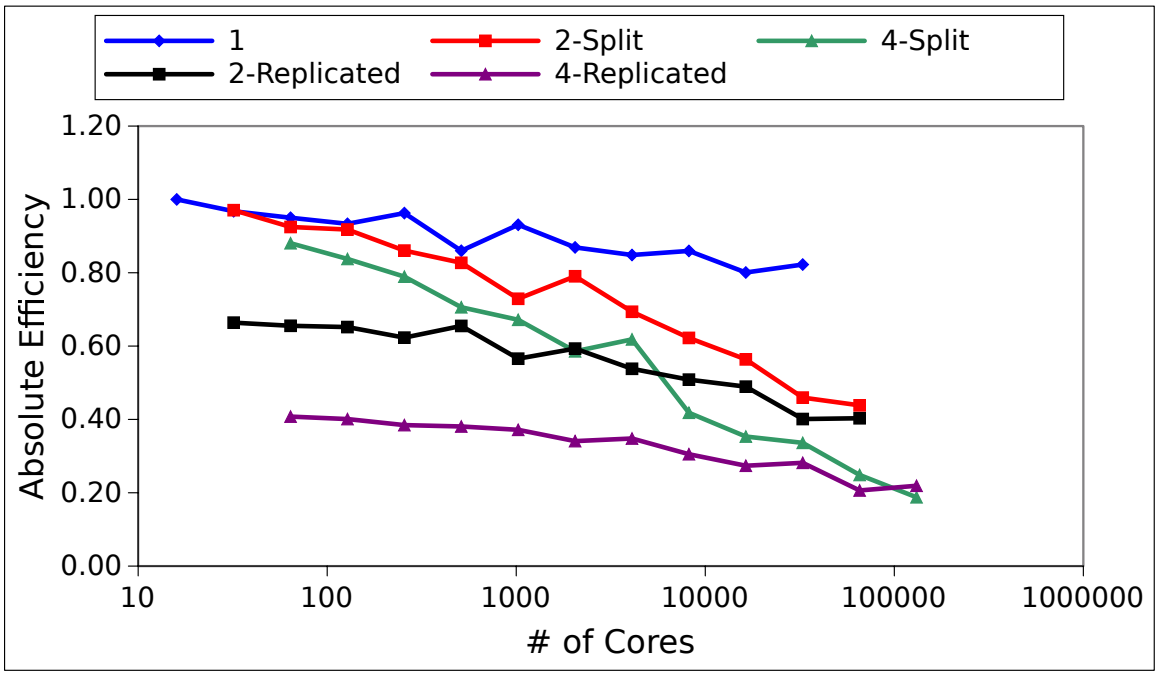

Figure 5.32: **Weak scaling absolute efficiency relative to 16-core 1-set base case.**

As in the strong scaling case, splitting and replicating histories across sets had effectively the same results for the weak scaling study with the iterations to converge and computed algorithmic efficiencies given by Table 5.6. Again, when the implementation efficiencies are computed, we see in Figure 5.33 that splitting histories among sets is still a more efficient manner of introducing multiple sets rather than replicating histories in an attempt to reduce the number of iterations required to converge. Here we also note that this is not a true weak scaling study as indicated by Eq (5.29) but rather an odd combination of both weak and strong scaling. Regardless of this, although multiple sets are not the most efficient use of resources in this analysis, they do enhance the time to solution through replication and do so significantly at core counts of O(1,000) and lower. In both the weak and strong scaling cases, due to the additional computational and communication overhead of adding sets to the system, one should never expect to seriously boost parallel efficiencies for MCSA using this technique except for certain instances past the strong scaling wall in a purely strong scaling environment. In a weak scaling environment, although no efficiency increase will be noted, one can expect an enhancement of time to convergence by replicating.

Case	Iterations	η_{alg}
1	22	1.0
2-split	22	1.0
2-replicated	16	1.375
4-split	22	1.0
4-replicated	13	1.692

Table 5.6: **Weak scaling algorithmic efficiency using multiple sets.** *Replicating histories improves algorithmic efficiency by converging in fewer iterations.*

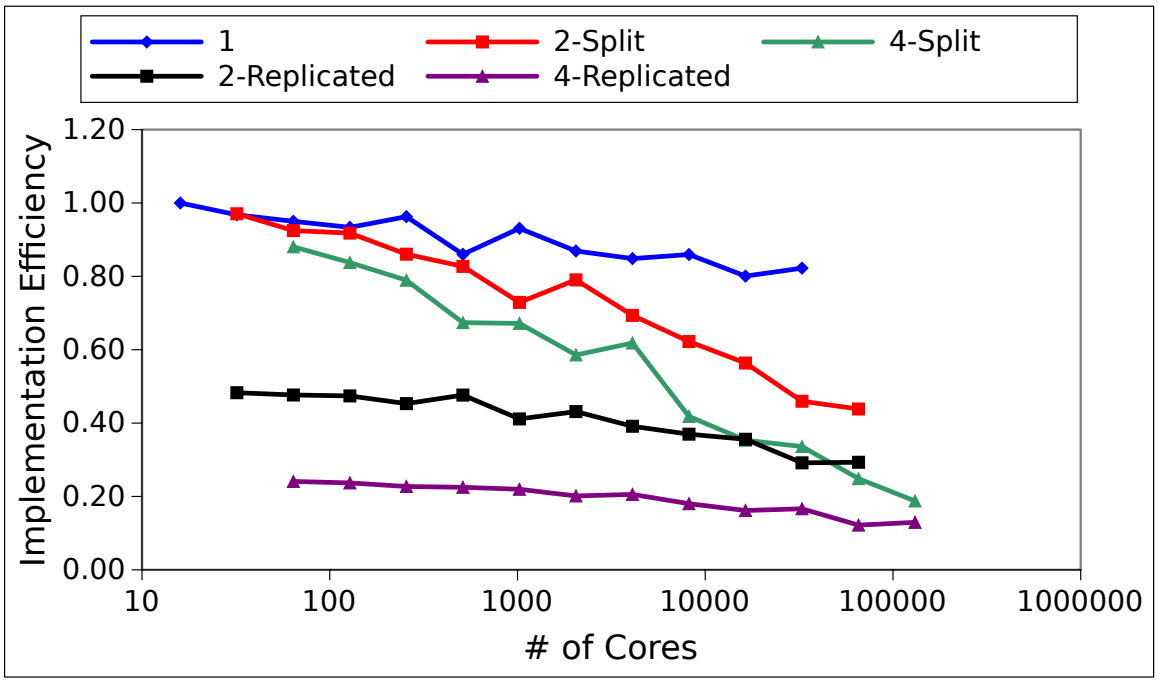

Figure 5.33: **Implementation parallel efficiency relative to 16-core 1-set base case.**

5.8.6 Subdomain Neumann-Ulam

Based on the analysis of the pure domain decomposed MCSA algorithm and subsequently the multiple sets and overlapping domain implementation, one may ask then ask if there is any need to communicate histories from domain-to-domain at all. Overlap was not as successful because because the additional communication required to reduce the overlapping tally vector eventually becomes larger than the communication savings during Monte Carlo transport. In addition, for local domains that are large enough to get good performance (e.g. those used in the weak scaling studies), then our analytic relationships for domain leakage, and in particular Eq (5.21), tell us that only about 5% of the histories in this diffusion problem will leave the domain in the first place. If that is true, then perhaps without any overlap or communication, just doing local Monte Carlo transport such that histories are terminated upon reaching the domain boundary may be satisfactory for accelerating the MCSA iteration. In this case, histories near the boundaries of the domain may be truncated before reaching the weight cutoff if they reach the domain boundary first. This truncation has the potential to increase the stochastic uncertainty of the Neumann-Ulam result and therefore increase the number of MCSA iterations required to converge. In addition, the scalability of the method will then be purely limited by the parallel matrix and vector operations required to implement the remainder of the algorithm on the subset of processors containing the linear problem.

Considering using MCSA in this way such that the Monte Carlo solve occurs only in the subdomains casts the method as more of a true domain decomposition as outlined by Chapter 14 of Saad's text on iterative methods (Saad, 2003). Compared to traditional domain decomposition methods we see many similarities including the potential to overlap the subdomains to improve the results. By building the boundary data as outlined in § 5.3.1, we are effectively gathering one level of the adjacent domains, albeit in this case to satisfy the data requirements of the Monte Carlo estimators. Furthermore, the fact that we are using residual Monte Carlo to compute the correction vector casts MCSA as a stochastic realization of an additive Schwarz method with a fixed point iteration used as a relaxation step[3]. Both the fixed point iteration and the computation of the residual propagate the solution from individual subdomains to the others through the matrix-vector multiply operation after all subdomain Monte Carlo solves have been completed. Perhaps an even more important consequence of viewing MCSA as a stochastic additive Schwarz procedure

[3]See section 14.3.3 in (Saad, 2003) for a full definition of the additive Schwarz method.

is that it has the potential to more readily fit as an accelerator into other linear solver techniques including GMRES or potentially as a preconditioner.

Using the Neumann-Ulam solver to compute the MCSA correction only in the subdomains, the strong and weak scaling studies were repeated to assess the performance gains made by eliminating communication in the Monte Carlo solver. For each scaling study, the 16-core base case with Monte Carlo communication was used as the reference point for efficiency computations in order to measure how eliminating communication boosts efficiencies. Figure 5.34 gives the results of the strong scaling study and Figure 5.35 gives the results of the the weak scaling study in terms of absolute efficiencies. In both cases, the benefits to parallel performance of using MCSA as a subdomain solver are obvious. In the strong scaling case we make the important observation that we hit the strong scaling wall at the same time with and without the additional Monte Carlo communication. Although the Monte Carlo communication did contribute to the efficiency decrease, it is clear that the remaining matrix and vector operations are the primary impediment to scalability. Furthermore, we see an even greater rise above unity in efficiencies because there is less communication overhead hiding the better cache utilization as the local problem size shrinks. At 16,384 cores, performing the subdomain solves boost the absolute efficiency from 38% to 67% relative to the reference case.

In the weak scaling study, we do not have the scaling wall to contend with and eliminating Monte Carlo communication through subdomain solves provides an effectively linear boost in the weak scaling absolute efficiency. We still see a small reduction in weak scaling efficiency as a function of core count due to the communication overhead for the parallel matrix and vector operations. Here we observe super unitary efficiencies for the subdomain case as there was a substantial improvement in runtimes relative to the 16-core base case with Monte Carlo communication.

Of additional concern for the subdomain method is the potential decrease in iterative performance due to the fact that histories near the boundary will potentially have their random walk truncated, thus decreasing the information available to compute the correction vector. For both the weak and strong scaling study, all calculations with split histories converged in 22 or in only a few cases 21 iterations for both the subdomain solve and the solve with communication giving an algorithmic efficiency of 1.048 for all cases. This small improvement in iteration count was not observed to be a function of problem size, number of processors, or whether the subdomain method was used. For the replicated history cases, the algorithmic efficiency improvement was not as good as when communication was allowed for the Monte Carlo algorithm. For

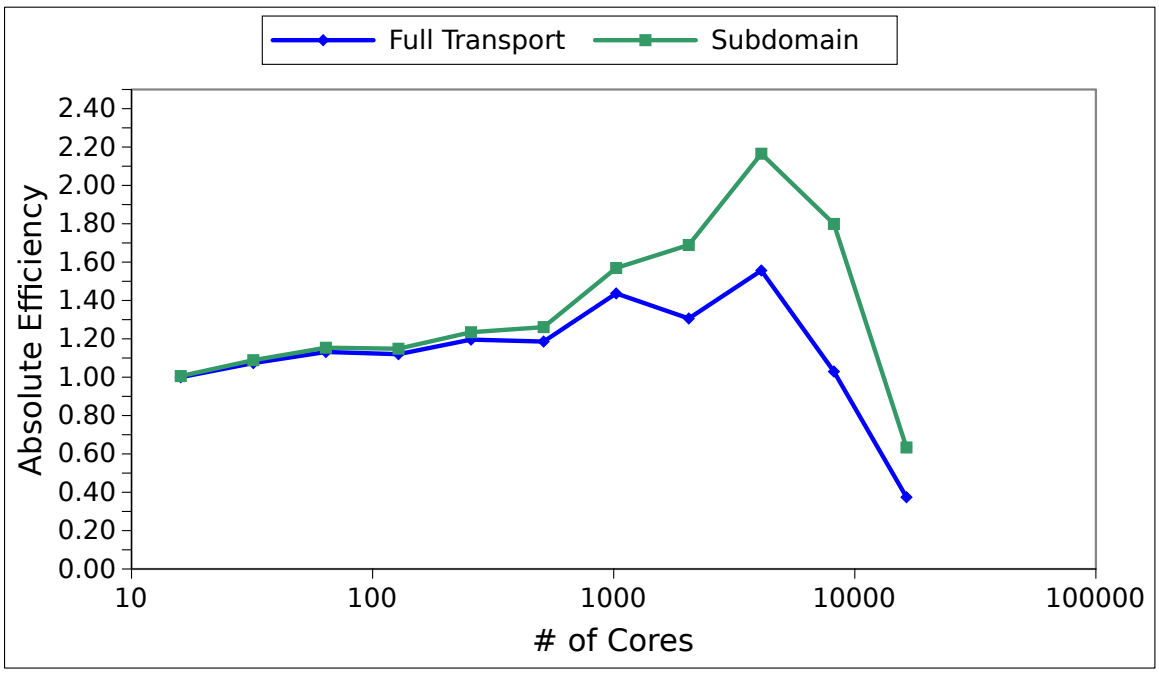

Figure 5.34: **Strong scaling absolute efficiency for pure domain decomposition.** *The subdomain Monte Carlo solver improves the parallel efficiency and therefore time to solution by eliminating all parallel communication during the Monte Carlo solve.*

these calculations, the number of iterations to converge was reduced to 18 or 19 in both the strong and weak scaling studies with lower core counts converging in fewer iterations.

Therefore, there is no reason to expect overlap to improve the iterative or scaling performance of the method as iterative performance is maintained without it for this particular problem. Overlap is not as effective because histories relinquish most of their useful information within the first few steps (which is why we can use such a large weight cutoff and there is an insensitivity to weight cutoff as shown in Figure 2.13). Keeping them around longer with overlap to do extra work in the local domain will not improve iterative performance and thus increase compute times.

As a final analysis of MCSA using subdomain Monte Carlo, the multiple sets scaling studies are repeated to determine if any performance benefits are made from replication by eliminating Monte Carlo communication. Figure 5.36 gives the runtimes for the strong scaling calculation and Figure 5.37 the computed absolute efficiencies. For the weak scaling studies, Figure 5.38 gives the wall times and Figure 5.39 the efficiencies. For both scaling studies, the results are effectively the same as for the

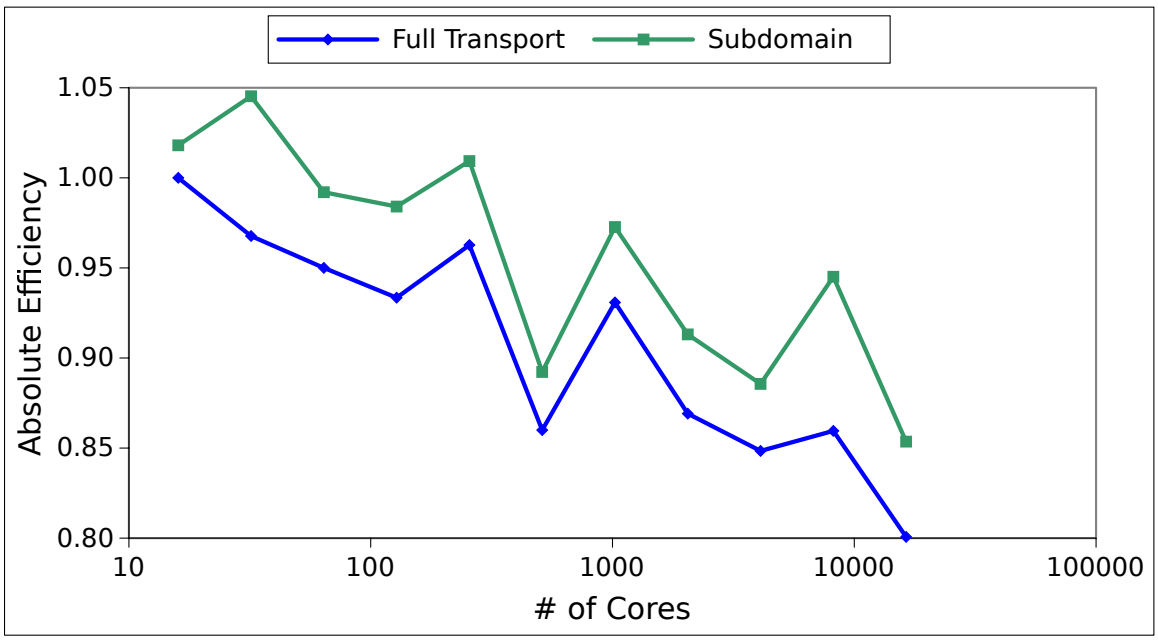

Figure 5.35: **Weak scaling absolute efficiency for pure domain decomposition.** *The subdomain Monte Carlo solver improves the parallel efficiency and therefore time to solution by eliminating all parallel communication during the Monte Carlo solve.*

case with Monte Carlo communication. This is again due to the fact that adding sets to the problem is not only an exercise in strong scaling but it also adds additional parallel overhead in the form of the tally vector reduction across blocks.

From a parallel performance perspective, MCSA may then be best used in a mode where the Monte Carlo is performed over only the subdomains with no replication or overlap added to the system. In this mode, iterative performance is maintained relative to the solutions with communication even for more ill-conditioned problems with larger spectral radii. Furthermore, no overlap was required to maintain iterative performance be letting histories near the boundary take a few more steps in their random walks. If in the future there are ill-conditioned problems where using subdomain Monte Carlo reduces the algorithmic efficiency of MCSA due to the information lost in the correction vector, overlap would be a viable mechanism to potentially maintain that iterative performance, keeping in mind the potential impacts on scalability that were observed in this work.

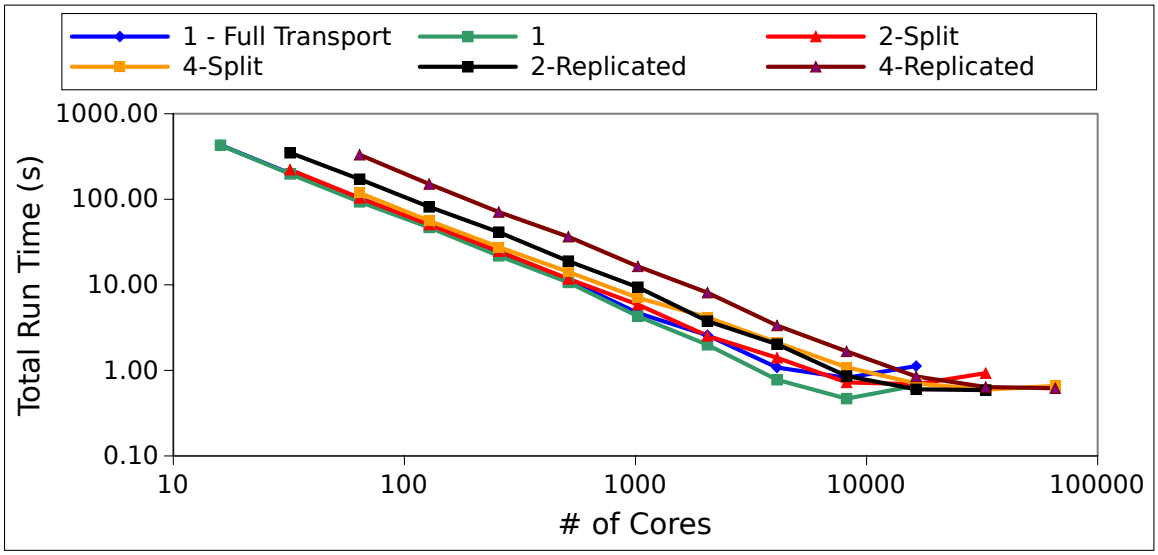

Figure 5.36: **Strong scaling total wall time with multiple sets and subdomain Monte Carlo.** *The subdomain Monte Carlo solver improves the parallel efficiency and therefore time to solution by eliminating all parallel communication during the Monte Carlo solve.*

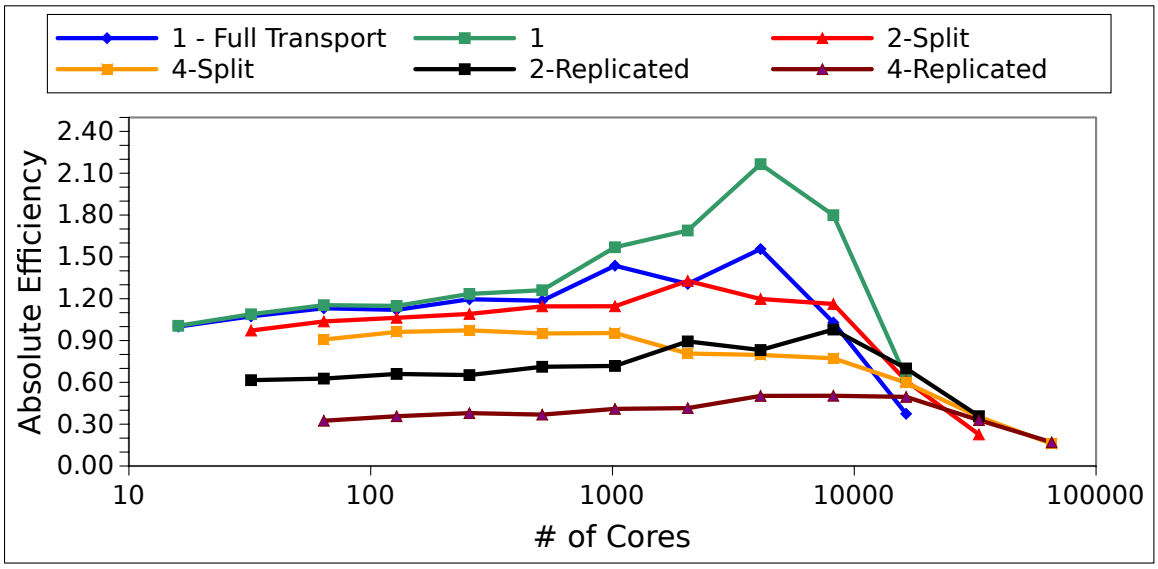

Figure 5.37: **Strong scaling absolute efficiency for with multiple sets and subdomain Monte Carlo.** *The subdomain Monte Carlo solver improves the parallel efficiency and therefore time to solution by eliminating all parallel communication during the Monte Carlo solve.*

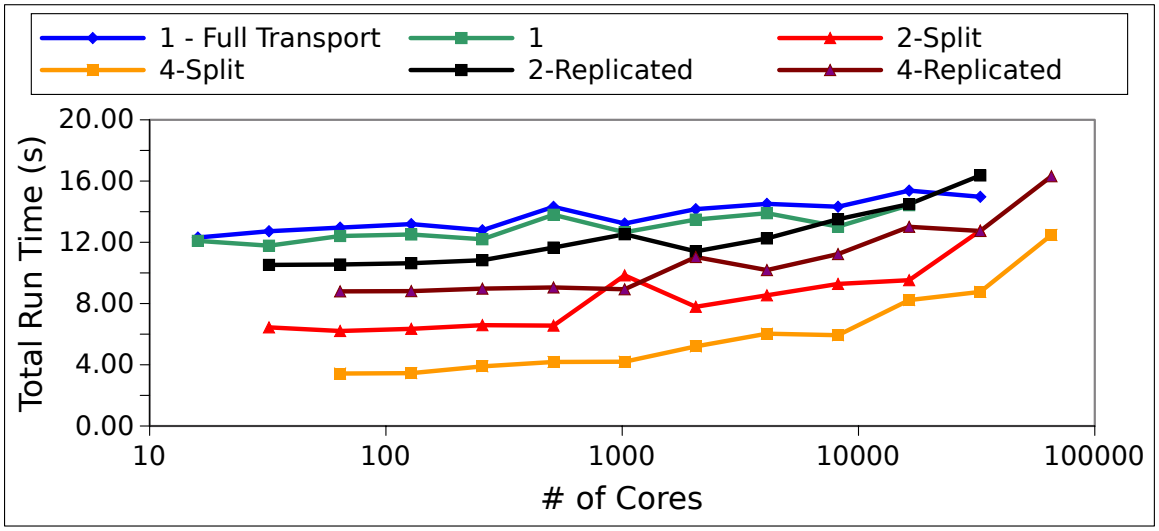

Figure 5.38: **Weak scaling total wall time with multiple sets and subdomain Monte Carlo.** *The subdomain Monte Carlo solver improves the parallel efficiency and therefore time to solution by eliminating all parallel communication during the Monte Carlo solve.*

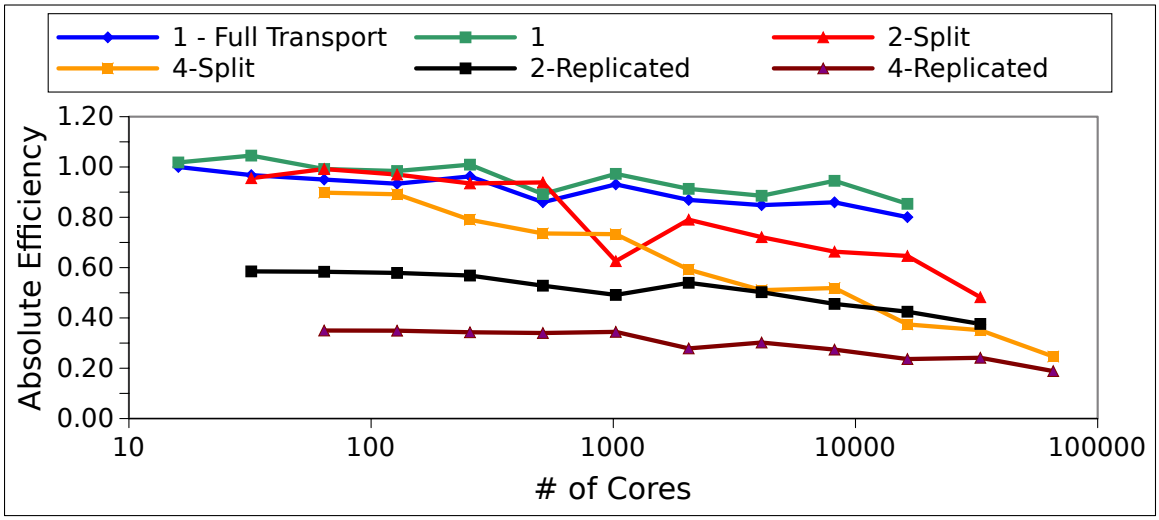

Figure 5.39: **Weak scaling absolute efficiency for with multiple sets and subdomain Monte Carlo.** *The subdomain Monte Carlo solver improves the parallel efficiency and therefore time to solution by eliminating all parallel communication during the Monte Carlo solve.*

Chapter 6

Conclusions and Analysis

Put conclusions here.

6.1 Monte Carlo Synthetic Acceleration Solutions for the SP_N Equations

6.2 Monte Carlo Synthetic Acceleration Solutions for Navier-Stokes Equations

6.3 Parallel Monte Carlo Synthetic Acceleration

6.4 Future Work

6.5 Closing Remarks

Appendix A

Conventional Solution Methods for Linear Systems

The discretization of partial differential equations (*PDEs*) through common methods such as finite differences (LeVeque, 2007), finite volumes (LeVeque, 2002), and finite elements (Zienkiewicz et al., 2005) ultimately generates sets of coupled equations in the form of matrix problems. In many cases, these matrices are sparse, meaning that the vast majority of their constituent elements are zero. This sparsity is due to the fact that the influence of a particular grid element only expands as far as a few of its nearest neighbors depending on the order of discretization used and therefore coupling among variables in a particular discrete equation in the system leads to a few non-zero entries. Because of the natural occurrence of sparse matrices in common numerical methods many iterative techniques have been developed to solve such systems. We discuss here conventional stationary and projection methods for solving sparse systems to provide the necessary background for the remainder of this work. Details on the parallelization of conventional methods are discussed.[1]

A.1 Stationary Methods

Stationary methods for linear systems arise from splitting the operator in Eq (2.1):

$$\mathbf{A} = \mathbf{M} - \mathbf{N}\,, \tag{A.1}$$

where the choice of $\mathbf{M}$ and $\mathbf{N}$ will be dictated by the particular method chosen. Using this split definition of the operator we can then write:

$$\mathbf{Mx} - \mathbf{Nx} = \mathbf{b}\,. \tag{A.2}$$

By rearranging, we can generate a form more useful for analysis:

$$\mathbf{x} = \mathbf{Hx} + \mathbf{c}\,, \tag{A.3}$$

[1]The contents of this appendix, particularly those sections relating to projection methods and matrix analysis, are heavily based on Saad's text (Saad, 2003).

where $\mathbf{H} = \mathbf{M}^{-1}\mathbf{N}$ is defined as the *iteration matrix* and $\mathbf{c} = \mathbf{M}^{-1}\mathbf{b}$. With the solution vector on both the left and right hand sides, an iterative method can then be formed:

$$\mathbf{x}^{k+1} = \mathbf{H}\mathbf{x}^k + \mathbf{c}\,, \tag{A.4}$$

with $k \in \mathbb{Z}^+$ defined as the *iteration index*. In general, we will define methods in the form of Eq (A.4) as *stationary methods*. Given this, we can then generate a few statements regarding the convergence of such stationary methods. Defining $\mathbf{e}^k = \mathbf{u}^k - \mathbf{u}$ as the solution error at the k^{th} iterate, we can subtract Eq (A.3) from Eq (A.4) to arrive at an error form of the linear problem:

$$\mathbf{e}^{k+1} = \mathbf{H}\mathbf{e}^k\,. \tag{A.5}$$

Our error after k iterations is then:

$$\mathbf{e}^k = \mathbf{H}^k\mathbf{e}^0\,. \tag{A.6}$$

In other words, successive application of the iteration matrix is the mechanism driving down the error in a stationary method. We can then place restrictions on the iteration matrix by using the tools developed in § (2.1). By assuming $\mathbf{H}$ is diagonalizable[2] (Saad, 2003), we then have:

$$\mathbf{e}^k = \mathbf{R}\mathbf{\Lambda}^k\mathbf{R}^{-1}\mathbf{e}^0\,, \tag{A.7}$$

where $\mathbf{\Lambda}$ contains the Eigenvalues of $\mathbf{H}$ on its diagonal and the columns of $\mathbf{R}$ contain the Eigenvectors of $\mathbf{H}$. Computing the 2-norm of the above form then gives:

$$||\mathbf{e}^k||_2 \leq ||\mathbf{\Lambda}^k||_2\ ||\mathbf{R}||_2\ ||\mathbf{R}^{-1}||_2\ ||\mathbf{e}^0||_2\,, \tag{A.8}$$

which gives:

$$||\mathbf{e}^k||_2 \leq \rho(\mathbf{H})^k\kappa(\mathbf{R})||\mathbf{e}^0||_2\,. \tag{A.9}$$

For iteration matrices where the Eigenvectors are orthogonal, $\kappa(\mathbf{R}) = 1$ and the error bound reduces to:

$$||\mathbf{e}^k||_2 \leq \rho(\mathbf{H})^k||\mathbf{e}^0||_2\,. \tag{A.10}$$

We can now restrict $\mathbf{H}$ by asserting that $\rho(\mathbf{H}) < 1$ for a stationary method to converge such that k applications of the iteration matrix will not cause the error to grow in Eq (A.10).

[2]We may generalize this to non-diagonalizable matrices with the Jordan canonical form of $\mathbf{H}$.

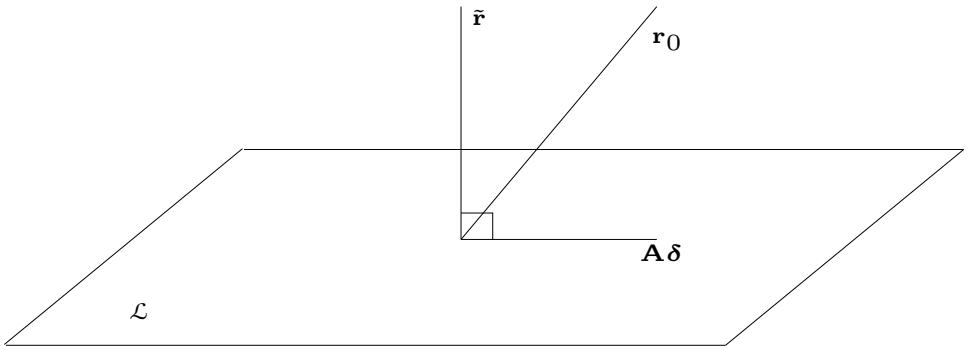

Figure A.1: **Orthogonality constraint of the new residual with respect to $\mathcal{L}$.** *By projecting $\mathbf{r}_0$ onto the constraint subspace, we minimize the new residual by removing those components.*

A.2 Projection Methods

Among the most common iterative methods used in scientific computing today for sparse systems are of a broad class known as *projection methods*. These methods not only provide access to more powerful means of reaching a solution, but also a powerful means of encapsulating the majority of common iterative methods including the stationary methods just discussed in a common mathematical framework. All projection methods are built around a core structure where the solution to Eq (2.1) is extracted from a *search subspace* $\mathcal{K}$ and bound by a *constraint subspace* $\mathcal{L}$ that will vary in definition depending on the iterative method selected. We build the approximate solution $\tilde{\mathbf{x}}$ by starting with an initial guess $\mathbf{x}_0$ and extracting a correction $\boldsymbol{\delta}$ from $\mathcal{K}$ such that:

$$\tilde{\mathbf{x}} = \mathbf{x}_0 + \boldsymbol{\delta},\ \boldsymbol{\delta} \in \mathcal{K}\,. \tag{A.11}$$

We bound this correction by asserting that the new residual, $\tilde{\mathbf{r}}$, be orthogonal to $\mathcal{L}$:

$$\langle \tilde{\mathbf{r}}, \mathbf{w} \rangle = 0,\ \forall \mathbf{w} \in \mathcal{L}\,. \tag{A.12}$$

We can generate a more physical and geometric-based understanding of these constraints by writing the new residual as $\tilde{\mathbf{r}} = \mathbf{r}_0 - \mathbf{A}\boldsymbol{\delta}$ and again asserting the residual must be orthogonal to $\mathcal{L}$. If $\tilde{\mathbf{r}}$ is to be orthogonal to $\mathcal{L}$, then $\mathbf{A}\boldsymbol{\delta}$ must be the projection of $\mathbf{r}_0$ onto the subspace $\mathcal{L}$ that eliminates the components of the residual that exist in $\mathcal{L}$. This situation is geometrically presented in Figure A.1.

From Figure A.1 we then note that the following geometric condition must hold:

$$||\tilde{\mathbf{r}}||_2 \leq ||\mathbf{r}_0||_2, \ \forall \mathbf{r}_0 \in \mathbb{R}^N \, , \tag{A.13}$$

meaning that the residual of the system will always be *minimized* with respect to the constraints. Given this minimization condition for the residual, we can form the outline of an iterative projection method. Consider a matrix $\mathbf{V}$ to form a basis of $\mathcal{K}$ and a matrix $\mathbf{W}$ to form a basis of $\mathcal{L}$. As $\boldsymbol{\delta} \in \mathcal{K}$ by definition in Eq (A.11), then $\boldsymbol{\delta}$ can instead be rewritten as:

$$\boldsymbol{\delta} = \mathbf{V}\mathbf{y}, \ \forall \mathbf{y} \in \mathbb{R}^N : \tag{A.14}$$

where $\mathbf{V}$ *projects* $\mathbf{y}$ onto $\mathcal{K}$. From the orthogonality constraint in Eq (A.12) it then follows that:

$$\mathbf{y} = (\mathbf{W}^T\mathbf{A}\mathbf{V})^{-1}\mathbf{W}^T\mathbf{r}_0 \, , \tag{A.15}$$

where here the projection onto $\mathcal{K}$ is constrained by the projection onto $\mathcal{L}$. Knowing this, we can then outline the following iteration scheme for a projection method:

$$\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k \, , \tag{A.16a}$$

$$\mathbf{y}^k = (\mathbf{W}^T\mathbf{A}\mathbf{V})^{-1}\mathbf{W}^T\mathbf{r}^k \, , \tag{A.16b}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{V}\mathbf{y}^k \, , \tag{A.16c}$$

where $\mathbf{V}$ and $\mathbf{W}$ are generated from the definitions of $\mathcal{K}$ and $\mathcal{L}$ and are updated prior to each iteration.

From an iteration standpoint, as we choose $\boldsymbol{\delta}$ from $\mathcal{K}$ and constrain it with $\mathcal{L}$, each iteration performs a projection that systematically annihilates the components of the residual that exists in $\mathcal{L}$. This then means that if our convergence criteria for an iterative method is bound to the residual of the system, then Eq (A.13) tells us that each projection step guarantees us that the norm of the new residual will never be worse than that of the previous step and will typically move us towards convergence. Depending on the qualities of the system in Eq (2.1), the selection of the subspaces $\mathcal{K}$ and $\mathcal{L}$ can serve to both guarantee convergence and optimize the rate at which the residual is decreased.

A.2.1 Krylov Subspace Methods

Among the most common projection techniques used in practice are a class of methods known as *Krylov subspace methods*. Here, the search subspace is defined as the *Krylov subspace*:

$$\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0) = span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{m-1}\mathbf{r}_0\} \,, \tag{A.17}$$

where m denotes the dimensionality of the subspace. In order to accommodate a more general structure for the operator in Eq (2.1), we often choose an *oblique* projection method where $\mathcal{K} \neq \mathcal{L}$. If we choose $\mathcal{L} = \mathbf{A}\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0)$, then we are ultimately solving the normal system $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ where $\mathbf{A}^T\mathbf{A}$ will be symmetric positive definite if $\mathbf{A}$ is nonsingular, thereby expanding the range of operators over which these methods are valid. This choice of constraint subspace also then gives us the result via Eq (A.12) that the residual is minimized for all $\boldsymbol{\delta} \in \mathcal{K}$, forming the basis for the *generalized minimum residual method* (GMRES) (Saad and Schultz, 1986).

Choosing GMRES as our model Krylov method, we are first tasked with finding a projector onto the subspace. We seek an orthonormal basis for $\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0)$ by an orthogonalization procedure that is commonly based on, but not limited to, the *Arnoldi* recurrence relation. The Arnoldi procedure will generate an orthonormal basis, $\mathbf{V}_m \in \mathbb{R}^{N\times m}$, via a variant of the Gram-Schmidt procedure that re-applies the operator for each consecutive vector, thus forming a basis that spans the subspace in Eq (A.17). Due to its equivalent dimensionality, m, to that of the subspace, we will refer to such recurrence relations as *long recurrence relations*. Those orthogonal projection procedures that have a dimensionality less than m will be referred to as *short recurrence relations*. Once $\mathbf{V}_m$ is found, per the constraint subspace definition it then follows that its basis is defined as $\mathbf{W}_m = \mathbf{A}\mathbf{V}_m$. Knowing the projections onto the search and constraint subspaces, the GMRES iteration may be formulated as follows:

We note here several properties of this formulation and how they may facilitate or hinder the solution of large-scale, sparse linear problems, also noting that these properties are common among many Krylov methods. First, from a memory perspective GMRES is efficient in that the operator $\mathbf{A}$ need not be explicitly stored. Rather, only the ability to compute the action of that operator on a vector of valid size is required. However, these savings in memory are balanced by the fact that the long recurrence relations used in the Arnoldi procedure require all vectors that span the Krylov space to be stored. If the size of these vectors becomes prohibitive, the Arnoldi procedure can be restarted at the cost of losing information in the orthogonalization process,

Algorithm A.1 GMRES Iteration

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\beta := ||\mathbf{r}_0||_2$
$\mathbf{v}_1 := \mathbf{r}_0/\beta$ ▷ Create the orthonormal basis for the Krylov subspace
for $j = 1, 2, \cdots, m$ **do**
  $h_{ij} \leftarrow \langle w_j, v_j \rangle$
  $w_j \leftarrow w_j - h_{ij} v_i$
end for
$h_{j+1,j} \leftarrow ||w_j||_2$
$v_{j+1} \leftarrow w_j / h_{j+1,j}$ ▷ Apply the orthogonality constraints
$\mathbf{y}_m \leftarrow argmin_y ||\beta \mathbf{e}_1 - \mathbf{H}_m \mathbf{y}||_2$
$\mathbf{x}_m \leftarrow \mathbf{x}_0 + \mathbf{V_m} \mathbf{y}_m$

creating the potential to generate new search directions that are not orthogonal to all previous search directions (and therefore less than optimal). From an implementation perspective, because the operator is not required to be formed, GMRES is significantly more flexible in its usage in that there are many instances where various processes serve to provide the action of that operator (e.g. radiation transport sweeps (Evans et al., 2010)) that normally may not be amenable to its full construction. In addition, the minimization problem is a straight-forward least-squares problem where **H** is an upper-Hessenberg matrix.

A.3 Parallel Projection Methods

Modern parallel implementations of projection methods on distributed memory architectures rely heavily on capabilities provided by general linear algebra frameworks. For methods like GMRES, this arises from the fact that Krylov methods require only a handful of operation types in their implementation that can be efficiently programmed on these architectures. Per Saad's text (Saad, 2003) and as noted in Algorithm A.1, these operations are preconditioning, matrix-vector multiplications, vector updates, and inner products. For the last three items, linear algebra libraries such as PETSc (Gropp and Smith, 1993) and Trilinos (Heroux et al., 2005) provide efficient parallel implementations for these operations. Depending on the type of preconditioning used, efficient parallel implementations may also be available for those operations. Due to their prevalence in modern numerical methods, parallel formulations these operations have warranted intense study (Tuminaro et al., 1998). In all cases, a series of scatter/-gather operations are required such that global communication operations must occur. Although the relative performance of such operations is bound to the implementation,

asymptotically performance should be the same across all implementations.

We will look at the three primary parallel matrix/vector operations as preconditioning is not an immediate requirement for implementing the algorithms. We note here that variants are available that reduce the number of global communications required (consider (Sosonkina et al., 1998) as an example of reducing global operation counts using a different orthogonalization procedure than Arnoldi), however, we will only consider the basic algorithms here as this handful of operations can be generalized to fit more complicated algorithms. In all of these cases, we assume a general matrix/vector formulation that is distributed in parallel such that both local and global knowledge of their decomposition is available on request. Furthermore, it is assumed that these objects are partitioned in such a way that the parallel formulation of the operator and vectors in Eq (2.1) will be such that each parallel process contains only a subset of the global problem and that subset forms a local set of complete equations. The form of this partitioning is problem dependent and often has a geometric or graph-based aspect to its construction in order to optimize communication patterns. Libraries such as Zoltan (Devine et al., 2002), provide implementations of such algorithms.

A.3.1 Parallel Vector Update

Parallel vector update operations arise from the construction of the orthonormal basis and the application of the correction generated by the constraints to the solution vector. Vector update operations are embarrassingly parallel in that they require no communication operations to be successfully completed; all data operated on is local. These operations are globally of the form:

$$\mathbf{y}[n] \leftarrow \mathbf{y}[n] + a * \mathbf{x}[n],\ \forall n \in [1, N_g]\,, \tag{A.18}$$

and locally of the form:

$$\mathbf{y}[n] \leftarrow \mathbf{y}[n] + a * \mathbf{x}[n],\ \forall n \in [1, N_l]\,, \tag{A.19}$$

where $\mathbf{y}$ and $\mathbf{x}$ are vectors of global size N_g, local size N_l, and $a \in \mathbb{R}^N$. In order to avoid communication, the vectors $\mathbf{y}$ and $\mathbf{x}$ must have the same parallel decomposition where each parallel process owns the same pieces of each vector.

A.3.2 Parallel Vector Product

Vector product operations are used in several instances during a Krylov iteration including vector norm computations and the orthogonalization procedure. By definition, the vector product is a global operation that effectively collapses a set of vectors to a single value. Therefore, we cannot eliminate all global communications. Instead, vector product operations are formulated as *global reduction operations* that are efficiently supported by modern message passing libraries. For the dot product of two vectors $\mathbf{y}$ and $\mathbf{x}$, a single reduction is required such that:

$$d_l = \mathbf{y}_l \cdot \mathbf{x}_l, \;\; d_g = \sum_p d_l \; , \tag{A.20}$$

where the l subscript denotes a local quantity, d_l is the local vector dot product, and d_g is the global dot product generated by summing the local dot products over all p processes. Parallel norm operations can be conducted with the same single reduction. Consider the infinity norm operation:

$$||x||_{\infty,l} = \max_n \mathbf{y}[n], \;\; \forall n \in [1, N_l] \tag{A.21a}$$

$$||x||_{\infty,g} = \max_p ||x||_{\infty,l} \; . \tag{A.21b}$$

In this form, the local infinity norm is computed over the local piece of the vector. The reduction operation is then formed over all p processes such that the global max of the vector is computed and distributed to all processes.

A.3.3 Parallel Matrix-Vector Multiplications

We finally consider parallel matrix-vector multiplication operations using sparse matrices in a compressed storage format by considering Saad's outline as well as the more formal work of Tuminaro (Tuminaro et al., 1998). For these operations, more complex communication patterns will be required given that the entire global vector is required in order to compute a single element of the local product vector. Fortunately, the vast majority of the global vector components will be multiplied by zero due to the sparsity of the matrix and therefore much of the vector can be neglected. Instead we only require data from a handful of other processes that can be acquired through asynchronous/synchronous communications. Consider the sparse matrix-vector multiply in Figure A.2 that is partitioned on 3 processors. Each process owns a set of equations that correlate to the physical domain of which it has ownership.

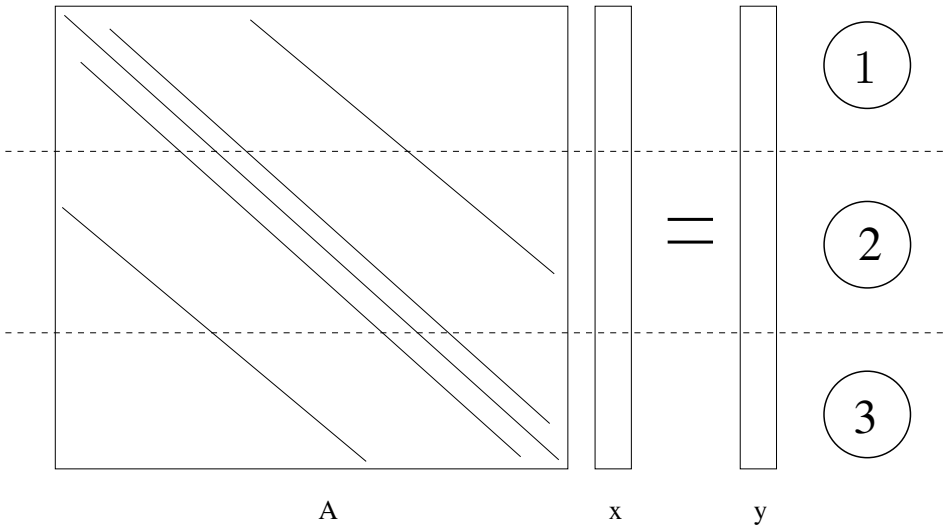

Figure A.2: **Sparse matrix-vector multiply $\mathbf{Ax} = \mathbf{y}$ operation partitioned on 3 processors.** *Each process owns a set of equations that correlates to its physical domain.*

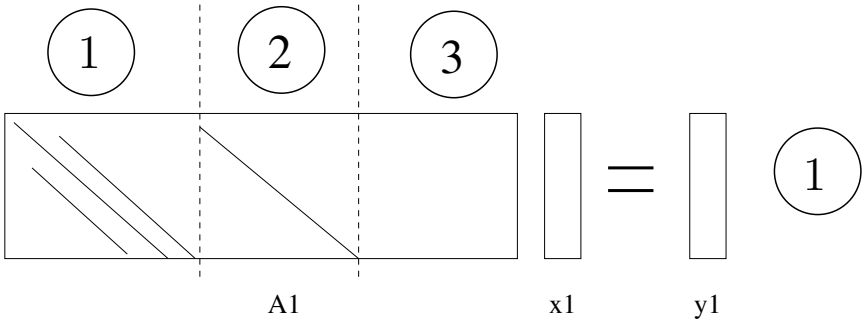

Figure A.3: **Components of sparse matrix-vector multiply operation owned by process 1.** *The numbers above the matrix columns indicate the process that owns the piece of the global vector they are acting on. In order to compute its local components of the matrix-vector product, process 1 needs its matrix elements along with all elements of the global vector owned by processes 2 and 3. The piece of the matrix shown is* $\mathbf{A}_1$ *and it is acting locally on* $\mathbf{x}_1$ *to compute the local piece of the product,* $\mathbf{y}_1$.

We can break down the equations owned by each process in order to devise an efficient scheme for the multiplication. Consider the portion of the matrix-vector multiply problem owned by process 1 in Figure A.2. As shown in Figure A.3, the components of the matrix will be multiplied by pieces of the vector that are owned by all processors. For those pieces of the matrix that are owned by process 1 that act on the vector

owned by process 1, we do these multiplications first as no communication is required. Next, process 1 gathers the components of the global vector owned by the other two processes that it requires to complete its part of the vector product. For this example, the components of matrix owned by process 1 that will operate on the global vector components owned by process 3 are zero, and therefore no vector elements are required to be scattered from process 3 to process 1. Those matrix elements owned by process 1 that will act on the piece of the vector owned by process 2 are not all non-zero, and therefore we must gather the entire process 2 vector components onto process 1 to complete the multiplication. Conversely, processes 2 and 3 must scatter their vector components that are required by other processes (such as process 1) in order to complete their pieces of the product. This then implies that these domain connections for proper gather and scatter combinations must be constructed a priori. These data structures are typically generated by a data partitioning library. Mathematically, if we are performing a global matrix-vector multiply of the form $\mathbf{Ax} = \mathbf{y}$, then for this example on process 1 we have a sequence of local matrix-vector multiplications:n $\mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_1\mathbf{x}_2 = \mathbf{y}_1$, with the subscripts provided by notation in Figure A.3. Here, some of the data is intrinsically local, and some must be gathered from other processes using the partitioning data structures.

A.3.4 Parallel Performance Implications for Krylov Methods

Knowing the parallel characteristics of the key operations we must perform in order to implement Krylov methods, we can make a few statements about parallel performance and implications for operation on machines of increasing size. Reconsider the matrix and vector operations required to implement Algorithm A.1. For very large distributed machines, the global reduction operations required at several levels of Krylov algorithms stand to reduce scalability and performance. In the case of GMRES, these reductions include vector norm operations and the inner products required for basis orthogonalization. Furthermore, communication between adjacent domains in matrix-vector multiply operations may also cause a bottleneck as the number of domains used in a simulation grows and the number of processors participating in the gather/scatter sequence requires a large communication bandwidth. The end result is that global data must be collected and communicated. For scaling improvement, we seek a reduction in these types of operations. In addition, these issues become more prominent as the Krylov iterations progress, causing the Krylov subspace to grow and

the total number of operations needed to orthogonalize that subspace to increase.

As an example of these performance implications in practice, in a 2001 work, Gropp and colleagues presented results on fluid dynamic simulations that heavily leveraged Krylov methods in their solution schemes (Gropp et al., 2001). In this follow-on to their 1997 Bell Prize-winning work, part of their analysis included identifying parallel scalability bottlenecks generated by solver implementations in a strong scaling exercise where the number of processors was increased with respect to a fixed global problem size. Gropp's observations show that for their particular hardware, a distributed memory machine similar to modern architectures, that global reduction operations did not impede scalability, meaning that the global reduction operation occupied approximately the same percentage of compute time independent of the number of processors used. Rather, it was the gather/scatter operations required to communicate data to neighboring processors that reduced performance with an increasing percentage of compute time consumed by these operations as processor count was increased. Furthermore, it was noted that this reduction in scaling was a product of poor algorithmic strong scaling rather than hardware or implementation related issues as the algorithm requires more data to be scattered/gathered as the number of processors and therefore computational domains increased. In the case of a weak scaling exercise, we would instead expect this percentage to exhibit a more desirable behavior of remaining constant for gather/scatter operations as problem size would be scaled with the number of processors.

Appendix B

Derivation of the P_N Equations

Here we derive the P_N equations, a simplified form of the general transport equation where Legendre polynomials are used to expand the angular flux and scattering cross section variables as a means of capturing the angular structure of the solution. Before deriving this form of the transport equation, we briefly discuss a few properties of Legendre polynomials that we will find useful in the derivation.

B.1 Legendre Polynomials

The Legendre polynomials are an orthogonal set of functions that are solutions to Legendre's differential equation. They have the following form (Lewis, 1993):

$$P_l(\mu) = \frac{1}{2^l l!}\frac{d^l}{d\mu^l}(\mu^2 - 1)^l \,. \tag{B.1}$$

These functions have several useful properties including orthogonality:

$$\int_{-1}^{1} P_l(\mu)P_{l'}(\mu)d\mu = \frac{1}{2l+1}\delta_{ll'} \,, \tag{B.2}$$

a recurrence relation:

$$\mu P_l(\mu) = \frac{1}{2l+1}[(l+1)P_{l+1}(\mu) + lP_{l-1}(\mu)] \,, \tag{B.3}$$

and an addition theorem:

$$P_l(\hat{\Omega}\cdot\hat{\Omega}') = \frac{1}{2n+1}\sum_{m=-l}^{l} Y_{lm}(\hat{\Omega})Y_{lm}^*(\hat{\Omega}') \,, \tag{B.4}$$

where the functions $Y_{lm}(\hat{\Omega})$ are the spherical harmonics. We can form the addition theorem in this way because the spherical harmonics are in fact just harmonic multiples of the Legendre polynomials:

$$Y_{lm}(\hat{\Omega}) = \sqrt{\frac{(2l+1)(l-m)!}{(l+m)!}}P_l^m(\mu)e^{i\omega m} \,, \tag{B.5}$$

where ω is the azimuthal component of the streaming direction. We can reduce Eq (B.4) for the planar geometry we are studying by ignoring the azimuthal components of the addition theorem. As shown in Eq (B.5), the azimuthal dependence is given by the harmonic component, $e^{i\omega m}$, and therefore we choose to ignore all terms in Eq. (B.4) where $m \neq 0$. This gives:

$$P_l(\hat{\Omega} \cdot \hat{\Omega}') = \frac{1}{2n+1} Y_{l0}(\hat{\Omega}) Y_{l0}^*(\hat{\Omega}') \,. \tag{B.6}$$

Per Eq (B.5) we have:

$$Y_{l0}(\hat{\Omega}) = \sqrt{2l+1} P_l^0(\mu) \,, \tag{B.7}$$

where $P_l^0(\mu) = P_l(\mu)$ is the 0^{th} associated Legendre function. Finally, we can reduce the addition theorem from Eq (B.6) with Eq (B.7) to a simple product for planar geometry:

$$P_l(\hat{\Omega} \cdot \hat{\Omega}') = P_l(\mu) P_l(\mu') \,. \tag{B.8}$$

B.2 Planar P_N Equations

With the Legendre polynomial properties defined above, we can proceed by deriving the P_N equations for planar geometry. For these equations, we will start by assuming a monoenergtic field of neutrons such that we are solving the following reduced form of the transport equation:

$$\mu \frac{\partial}{\partial x} \psi(x,\mu) + \sigma(x)\psi(x,\mu) = \int \sigma_s(x, \hat{\Omega}' \rightarrow \hat{\Omega}) \psi(\vec{r}, \hat{\Omega}') d\Omega' + \frac{q(x)}{4\pi} \,. \tag{B.9}$$

The P_N equations introduce the approximation that the angular dependence of the scattering cross section, σ_s, and the angular flux ψ, can be discretized by expanding them in Legendre polynomials as follows:

$$\psi(x,\mu) = \sum_{n=0}^{\infty} (2n+1) P_n(\mu) \phi_n(x) \,, \tag{B.10}$$

$$\sigma_{sm}(x) = \sum_{m=0}^{\infty} (2m+1) P_m(\mu) \sigma_s(x) \,, \tag{B.11}$$

where we have supressed the 2π generated by integrating away the azimuthal angular component and $\phi_n(x)$ in Eq (B.10) is referred to as the n^{th} Legendre moment of the

neutron flux and is given by:

$$\phi_n(x) = \int_{-1}^{1} P_n(\mu)\psi(x,\mu)d\mu \,. \tag{B.12}$$

We first insert the expansions given by Eq (B.10) and (B.11) into the planar transport equation given by Eq (B.9):

$$\frac{\partial}{\partial x}\Big[\sum_{n=0}^{\infty}(2n+1)\phi_n\mu P_n(\mu)\Big] + \sigma\sum_{n=0}^{\infty}(2n+1)\phi_n P_n(\mu) = \int_{-1}^{1}\sum_{m=0}^{\infty}(2m+1)\sigma_{sm}P_m(\mu_0)\sum_{n=0}^{\infty}(2n+1)\phi_n P_n(\mu')d\mu' + q \,, \tag{B.13}$$

where the dependence on the spatial variable, x, has been supressed. To arrive at the P_N equations we multiply Eq (B.13) by $P_m(\mu)$ and integrate over the angular domain $\int_{-1}^{1} d\mu$. We will look at each term in Eq (B.13) individually.

Streaming Term We first apply the multiplication and integration as prescribed above:

$$\frac{\partial}{\partial x}\left[\sum_{n=0}^{\infty}(2n+1)\mu\phi_n P_n(\mu)\right] \rightarrow \int_{-1}^{1}\frac{\partial}{\partial x}\left[\sum_{n=0}^{\infty}(2n+1)\phi_n\mu P_n(\mu)P_m(\mu)\right]d\mu \,. \tag{B.14}$$

The $\mu P_n(\mu)$ term can be eliminated via the recurrence relation given by Eq (B.3):

$$\int_{-1}^{1}\frac{\partial}{\partial x}\left[\sum_{n=0}^{\infty}(2n+1)\frac{\phi_n}{2n+1}[(n+1)P_{n+1}(\mu)+nP_{n-1}(\mu)]P_m(\mu)\right]d\mu \,, \tag{B.15}$$

which expands to:

$$\sum_{n=0}^{\infty}\frac{\partial}{\partial x}\phi_n\left[(n+1)\int_{-1}^{1}P_{n+1}(\mu)P_m(\mu)d\mu + n\int_{-1}^{1}P_{n-1}(\mu)P_m(\mu)d\mu\right] . \tag{B.16}$$

This reveals the orthogonality relation given by Eq (B.2) that when inserted into Eq (B.15):

$$\sum_{n=0}^{\infty}\frac{\partial}{\partial x}\phi_n\left[(n+1)\frac{1}{2n+1}\delta_{n,n+1} + n\frac{1}{2n+1}\delta_{n,n-1}\right] . \tag{B.17}$$

We can then distribute the Legendre moment to arrive at the final form of the streaming term:

$$\sum_{n=0}^{\infty}\frac{\partial}{\partial x}\frac{1}{2n+1}\Big[(n+1)\phi_{n+1} + n\phi_{n-1}\Big] . \tag{B.18}$$

Collision Term To reduce the collision term, the orthogonality relation is again applied after the integration:

$$\sigma \sum_{n=0}^{\infty}(2n+1)\phi_n P_n(\mu) \rightarrow \int_{-1}^{1} \sigma \sum_{n=0}^{\infty}(2n+1)\phi_n P_n(\mu)P_m(\mu)d\mu \,, \tag{B.19}$$

$$\sigma \sum_{n=0}^{\infty}(2n+1)\phi_n \frac{1}{2n+1} \,, \tag{B.20}$$

giving for the final collision term:

$$\sum_{n=0}^{\infty}\sigma\phi_n \,. \tag{B.21}$$

Scattering Term For the scattering term:

$$\int_{-1}^{1} \sum_{m=0}^{\infty}(2m+1)\sigma_{sm}P_m(\mu_0) \sum_{n=0}^{\infty}(2n+1)\phi_n P_n(\mu')d\mu' \rightarrow$$
$$\int_{-1}^{1}\int_{-1}^{1} \sum_{m=0}^{\infty}(2m+1)\sigma_{sm}P_m(\mu)P_m(\mu_0) \sum_{n=0}^{\infty}(2n+1)\phi_n P_n(\mu')d\mu' d\mu \,. \tag{B.22}$$

The addition thereom from Eq (B.8) is applied to give:

$$\int_{-1}^{1}\int_{-1}^{1} \sum_{m=0}^{\infty}(2m+1)\sigma_{sm}P_m(\mu)P_m(\mu)P_m(\mu') \sum_{n=0}^{\infty}(2n+1)\phi_n P_n(\mu')d\mu' d\mu \,, \tag{B.23}$$

which can be rearranged as:

$$\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}(2m+1)\sigma_{sm}(2n+1)\phi_n \int_{-1}^{1} P_m(\mu')P_n(\mu')d\mu' \int_{-1}^{1} P_m(\mu)P_m(\mu)d\mu \,. \tag{B.24}$$

Again we can apply orthogonality to eliminate the Legendre polynomials:

$$\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}(2m+1)\sigma_{sm}(2n+1)\phi_n \frac{1}{2n+1}\delta_{nm}\frac{1}{2m+1}\delta_{mm} \,, \tag{B.25}$$

which is reduced to:

$$\sum_{n=0}^{\infty}\sigma_{sn}\phi_n \,, \tag{B.26}$$

with the dependence on the index m eliminated.

Source Term The last term we are concerned with in Eq (B.13) is the source term:

$$q \to \int_{-1}^{1} qP_n(\mu)d\mu \,. \tag{B.27}$$

We can leverage orthogonality by multiplying by $P_0(\mu) = 1$:

$$\int_{-1}^{1} qP_0P_n(\mu)d\mu = \frac{q}{2*0+1}\delta_{n0} \,, \tag{B.28}$$

giving a final source term of:

$$q\delta_{n0} \,. \tag{B.29}$$

Now that we have expanded all angular dependent terms in Eq (B.13) and reduced them appropriately, we can combine them to generate the P_N equations:

$$\sum_{n=0}^{\infty} \frac{\partial}{\partial x}\frac{1}{2n+1}\Big[(n+1)\phi_{n+1} + n\phi_{n-1}\Big] + \sum_{n=0}^{\infty}\sigma\phi_n = \sum_{n=0}^{\infty}\sigma_{sn}\phi_n + q\delta_{n0} \,. \tag{B.30}$$

More formally, the P_N equations are written as:

$$\frac{1}{2n+1}\frac{\partial}{\partial x}\Big[(n+1)\phi_{n+1} + n\phi_{n-1}\Big] + \Sigma_n\phi_n = q\delta_{n0} \,, \tag{B.31}$$

where $\Sigma_n = \sigma - \sigma_{sn}$ and the summations are truncated at some level of approximation N such that $n = 0, 1, \ldots, N$. This yields a set of $N+1$ equations for $N+2$ flux moments. We therefore require an additional equation to close the system. In accordance with the series truncation as an approximation we choose the last moment in the expansion to be zero:

$$\phi_{N+1} = 0 \,. \tag{B.32}$$

As an example, we will construct the P5 equations from Eq (B.31) and the closure

given by Eq (B.32):

$$\frac{\partial}{\partial x}\phi_1 + \Sigma_0\phi_0 = q\,, \tag{B.33a}$$

$$\frac{1}{3}\frac{\partial}{\partial x}\Big[2\phi_2 + \phi_0\Big] + \Sigma_1\phi_1 = 0\,, \tag{B.33b}$$

$$\frac{1}{5}\frac{\partial}{\partial x}\Big[3\phi_3 + 2\phi_1\Big] + \Sigma_2\phi_2 = 0\,, \tag{B.33c}$$

$$\frac{1}{7}\frac{\partial}{\partial x}\Big[4\phi_4 + 3\phi_2\Big] + \Sigma_3\phi_3 = 0\,, \tag{B.33d}$$

$$\frac{1}{9}\frac{\partial}{\partial x}\Big[5\phi_5 + 4\phi_3\Big] + \Sigma_4\phi_4 = 0\,, \tag{B.33e}$$

$$\frac{1}{11}\frac{\partial}{\partial x}5\phi_4 + \Sigma_5\phi_5 = 0\,. \tag{B.33f}$$

This gives us a set of 6 coupled equations for the six Legendre moments requested defined over the entire spatial domain for a single energy group. In practice, only odd-numbered P_N orders are generally used (Lewis, 1993). This is due to the fact that using odd N yields an even number of $N+1$ equations which can be split evenly on the left and right boundaries of the problem to facilitate the description of boundary conditions. We will choose this convention when deriving the boundary conditions.

B.3 Boundary Conditions for the P_N Equations

Per the analysis in (Lewis, 1993), two types of boundary conditions will be discussed: reflecting and Marshak. Marshak boundary conditions can be used to specify both vacuum conditions and isotropic source conditions on the boundary.

Reflecting Boundary Conditions In this case, the incoming flux should be equivalent to the outgoing flux at the boundary point x_b:

$$\psi(x_b, \mu) = \psi(x_b, -\mu)\,. \tag{B.34}$$

Given the legendre expansion for the flux defined in Eq (B.10) and the Legendre polynomial property that $P_n(\mu) = (-1)^n P_n(-\mu)$, the condition specified by Eq (B.34) can be satisfied if

$$\phi_n = 0,\ \forall \text{ odd } n\,, \tag{B.35}$$

as all even n yield $P_n(\mu) = P_n(-\mu)$ and therefore an equivalent reflecting condition for the flux moments.

Marshak Boundary Conditions The Marshak conditions come directly from the Legendre moments of the flux:

$$\int_{\mu_b} P_i(\mu)\psi(\mu)d\mu = \int_{\mu_b} P_i(\mu)\psi_b(\mu)d\mu \qquad \text{for } i = 1, 3, ..., N\,, \tag{B.36}$$

where $\psi_b(\mu)$ is the prescribed angular flux on the boundary of interest and μ_b the angular domain defined by the boundary. To discretize this condition, we again insert the angular flux expansions from Eq (B.10) for the fluxes defined in the domain:

$$\int_{\mu_b} P_i(\mu) \sum_{n=0}^{N} (2n+1)\phi_n P_n(\mu)d\mu = \int_{\mu_b} P_i(\mu)\psi_b(\mu)d\mu\,. \tag{B.37}$$

The boundary flux, $\phi_b(\mu)$ is assumed to known and therefore Eq (B.37) defines a set of $(N+1)/2$ equations to be solved on each boundary in the planar case, closing the system in the spatial domain.

As an example of applying the Marshak conditions, consider the P_3 case with an isotropic boundary source ϕ_b on the left side of the domain. In this case, the angular domain over which the boundary flux is defined will be $\mu_b \in [0, 1]$, giving the bounds of integration. We first expand the summation for $i = 1$:

$$\int_0^1 \mu \left[\phi_0 + 3\phi_1\mu + \frac{5}{2}\phi_2(3\mu^2 - 1) + \frac{7}{2}\phi_3(5\mu^3 - 3\mu)\right] d\mu = \int_0^1 \mu\phi_b d\mu\,, \tag{B.38}$$

and then $i = 3$:

$$\int_0^1 \frac{1}{2}(5\mu^3 - 3\mu)\left[\phi_0 + 3\phi_1\mu + \frac{5}{2}\phi_2(3\mu^2 - 1) + \frac{7}{2}\phi_3(5\mu^3 - 3\mu)\right] d\mu = \int_0^1 \frac{1}{2}(5\mu^3 - 3\mu)\phi_b d\mu\,. \tag{B.39}$$

Expanding the polynomials in μ and carrying out the simple integration then gives 2 equations for the left hand boundary:

$$\phi_0 + 2\phi_1 + \frac{5}{4}\phi_2 = \phi_b\,, \tag{B.40}$$

$$\phi_0 - 5\phi_2 - 8\phi_3 = \phi_b\,. \tag{B.41}$$

The right hand side boundary condition will yield 2 complementary equations if Marshak conditions are used or 2 non-zero moments to be solved for if reflected conditions are used. The formulation above also holds for vacuum conditions where $\phi_b = 0$.

B.4 Eigenvalue Form of the P_N Equations

In addition to fixed source problems, the P_N equations may be modified to solve eigenvalue problems by applying the same discretization techniques to the fission source in Eq (3.3). Considering the multigroup case, we first replace the integration over energy with a summation over energy groups:

$$\frac{1}{k}\chi(E)\iint \nu\sigma_f(x,E')\psi(x,\hat{\Omega}',E')d\Omega' dE' = \frac{1}{k}\sum_0^G \chi^g \int \nu\sigma_f^{g'}(x)\psi^{g'}(x,\hat{\Omega}')d\Omega' \,, \quad \text{(B.42)}$$

where $G = N_g - 1$ and N_g is the number of energy groups. Next, we expand the angular flux in the fission term by applying the expansion in Eq (B.10) and remove the dependence on the azimuthal angle:

$$\frac{1}{k}\sum_0^G \chi^g \int \nu\sigma_f^{g'}(x)\psi^{g'}(x,\hat{\Omega}')d\Omega' = \frac{1}{k}\sum_0^G \chi^g \int_{-1}^{1} \nu\sigma_f^{g'}(x)\sum_{n=0}^{\infty}(2n+1)P_n(\mu)\phi_n^{g'}(x)d\mu' \,. \quad \text{(B.43)}$$

We can again use the the orthogonality property of the Legendre polynomials by multplying by $P_0(\mu) = 1$:

$$\frac{1}{k}\sum_0^G \chi^g \int_{-1}^{1} \nu\sigma_f^{g'}(x)\sum_{n=0}^{\infty}(2n+1)P_n(\mu)P_0(\mu)\phi_n^{g'}(x)d\mu' \,, \quad \text{(B.44)}$$

giving:

$$\frac{1}{k}\sum_0^G \chi^g \nu\sigma_f^{g'}(x)\phi_n^{g'}(x)\delta_{n0} \,, \quad \text{(B.45)}$$

where χ^g and $\sigma_f^{g'}\phi^{g'}$ means that an effective inner product amongst group terms will be computed with the discrete χ spectrum and the fission reaction rate. In a multigoup form, if we choose to represent the energy dependent moments as vectors, $\mathbf{\Phi_n}$, composed of all energy groups, then this inner product takes the form of a spatially

dependent fission matrix $\mathbf{F}$:

$$\mathbf{F} = \nu \begin{bmatrix} (\chi^0\sigma_f^0) & (\chi^0\sigma_f^1) & \cdots & (\chi^0\sigma_f^G) \\ (\chi^1\sigma_f^0) & (\chi^1\sigma_f^1) & \cdots & (\chi^1\sigma_f^G) \\ \vdots & \vdots & \ddots & \vdots \\ (\chi^G\sigma_f^0) & (\chi^G\sigma_f^1) & \cdots & (\chi^G\sigma_f^G) \end{bmatrix}. \tag{B.46}$$

This gives a final multigroup P_N fission source of:

$$\frac{1}{k}\mathbf{F}\mathbf{\Phi}_\mathbf{n}\delta_{n0} \tag{B.47}$$

where the spatial dependence of the flux moments and fission matrix has been supressed.

Appendix C

Derivation of the Multigroup SP_N Equations

In the text, we formulated the SP_N equations for a single neutron energy. To expand these equations for multiple energies, we start by stating the multigroup neutron transport equation for a single dimension in planar geometry:

$$\mu\frac{\partial}{\partial x}\psi^g(x,\mu)+\sigma^g(x)\psi^g(x,\mu)= \sum_{g'=0}^{G}\int\sigma_s^{gg'}(x,\hat{\Omega}'\rightarrow\hat{\Omega})\psi^{g'}(x,\hat{\Omega}')d\Omega'+\frac{q^g(x)}{4\pi}\,, \quad \text{(C.1)}$$

where g denotes the group index of 0 to G groups, $G=N_g-1$, and the integration of the scattering emission term over energy has been replaced by a discrete summation. For scattering, $\sigma_s^{gg'}$ provides the probability of scattering at a particular angle from group g to g'. The result is an equation nearly identical in form to Eq (3.2) where now instead of forming the SP_N equations for a single energy group, we form them for each of the energy groups with group coupling occurring through the scattering term. The multigroup P_N equations are then:

$$\frac{1}{2n+1}\frac{\partial}{\partial x}\Big[(n+1)\phi_{n+1}^g+n\phi_{n-1}^g\Big]+\sum_{g'}(\sigma^g\delta_{gg'}-\sigma_{sn}^{gg'})\phi_n^g=q\delta_{n0}\,, \quad \text{(C.2)}$$

for $n=0,1,\ldots,N$ where the flux and scattering moments are defined in a group. We observe that a $N_g\times N_g$ scattering matrix is formed:

$$\mathbf{\Sigma}_n=\sum_{g'}(\sigma^g\delta_{gg'}-\sigma_{sn}^{gg'})\,, \quad \text{(C.3)}$$

and when expanded gives:

$$\mathbf{\Sigma}_n = \begin{bmatrix} (\sigma^0 - \sigma_{sn}^{00}) & -\sigma_{sn}^{01} & \dots & -\sigma_{sn}^{0G} \\ -\sigma_{sn}^{10} & (\sigma^1 - \sigma_{sn}^{11}) & \dots & -\sigma_{sn}^{1G} \\ \vdots & \vdots & \ddots & \vdots \\ -\sigma_{sn}^{G0} & -\sigma_{sn}^{G1} & \dots & (\sigma^G - \sigma_{sn}^{GG}) \end{bmatrix} . \tag{C.4}$$

It is also useful to combine the group flux moments and sources into a single vector for more compact notation:

$$\mathbf{\Phi_n} = (\phi_n^0 \ \phi_n^1 \ \cdots \phi_n^G)^T , \tag{C.5}$$

$$\mathbf{q} = (q^0 \ q^1 \ \cdots q^G)^T . \tag{C.6}$$

Next, we apply the SP_N approximation to Eq (C.2) in identical fashion to the monoenergetic case. This gives:

$$\begin{aligned} -\nabla \cdot \Bigg[\frac{n}{2n+1} \mathbf{\Sigma_{n-1}}^{-1} \nabla \Big(\frac{n-1}{2n-1} \mathbf{\Phi_{n-2}} + \frac{n}{2n-1} \mathbf{\Phi_n} \Big) \\ + \frac{n+1}{2n+1} \mathbf{\Sigma_{n+1}}^{-1} \nabla \Big(\frac{n+1}{2n+3} \mathbf{\Phi_n} + \frac{n+2}{2n+3} \mathbf{\Phi_{n+2}} \Big) \Bigg] \\ + \mathbf{\Sigma_n \Phi_n} = \mathbf{q} \delta_{n0} \qquad n = 0, 2, 4, \cdots, N . \end{aligned} \tag{C.7}$$

This adds more complexity than the monoenergetic formulation in that all unknowns in this group of equations are now vector quantities and scattering relationships are contained in matrices rather than a scalar quantity. Because of this, the same sequence of variable changes and algebra can be used to build a set of matrix equations, this time in a block formulation:

$$-\nabla \cdot \mathbb{D}_n \nabla \mathbb{U}_n + \sum_{m=1}^{4} \mathbb{A}_{nm} \mathbb{U}_m = \mathbb{Q}_n , \tag{C.8}$$

where the definition of all quantities are the same with internal scalar values replaced by the group-vector values. In addition, $\mathbb{A}$ is now a block matrix of $N_g \times N_g$ sub-matrices

generated from the moment scattering matrices:

$$\mathbf{A} = \begin{bmatrix} (\boldsymbol{\Sigma}_0) & (-\frac{2}{3}\boldsymbol{\Sigma}_0) & (\frac{8}{15}\boldsymbol{\Sigma}_0) & (-\frac{16}{35}\boldsymbol{\Sigma}_0) \\ (-\frac{2}{3}\boldsymbol{\Sigma}_0) & (\frac{4}{9}\boldsymbol{\Sigma}_0 + \frac{5}{9}\boldsymbol{\Sigma}_2) & (-\frac{16}{45}\boldsymbol{\Sigma}_0 - \frac{4}{9}\boldsymbol{\Sigma}_2) & (\frac{32}{105}\boldsymbol{\Sigma}_0 + \frac{8}{21}\boldsymbol{\Sigma}_2) \\ (\frac{8}{15}\boldsymbol{\Sigma}_0) & (-\frac{16}{45}\boldsymbol{\Sigma}_0 - \frac{4}{9}\boldsymbol{\Sigma}_2) & (\frac{64}{225}\boldsymbol{\Sigma}_0 + \frac{16}{45}\boldsymbol{\Sigma}_2 + \frac{9}{25}\boldsymbol{\Sigma}_4) & (-\frac{128}{525}\boldsymbol{\Sigma}_0 - \frac{32}{105}\boldsymbol{\Sigma}_2 - \frac{54}{175}\boldsymbol{\Sigma}_4) \\ (-\frac{16}{35}\boldsymbol{\Sigma}_0) & (\frac{32}{105}\boldsymbol{\Sigma}_0 + \frac{8}{21}\boldsymbol{\Sigma}_2) & (-\frac{128}{525}\boldsymbol{\Sigma}_0 - \frac{32}{105}\boldsymbol{\Sigma}_2 - \frac{54}{175}\boldsymbol{\Sigma}_4) & (\frac{256}{1225}\boldsymbol{\Sigma}_0 + \frac{64}{245}\boldsymbol{\Sigma}_2 + \frac{324}{1225}\boldsymbol{\Sigma}_4 + \frac{13}{49}\boldsymbol{\Sigma}_6) \end{bmatrix}. \tag{C.9}$$

Appendix D

Boundary Conditions for the SP_N Equations

We approach the formulation of the boundary conditions in much the same way as for the P_N equations with both reflecting and Marshak conditions possible. To begin, we perform the expansion in Eq (B.37) for the left side of the planar system, this time for $N = 7$ to correspond to our SP_7 system. For $i = 1$:

$$\int_0^1 \mu \Bigg[\phi_0 + 3\phi_1\mu + \frac{5}{2}\phi_2(3\mu^2 - 1) + \frac{7}{2}\phi_3(5\mu^3 - 3\mu) + \frac{1}{8}(63\mu^5 - 70\mu^3 + 15\mu) \Bigg] d\mu = \int_0^1 \mu\phi_b d\mu \,, \quad \text{(D.1)}$$

for $i = 3$,

$$\int_0^1 \frac{1}{2}(5\mu^3 - 3\mu) \Bigg[\phi_0 + 3\phi_1\mu + \frac{5}{2}\phi_2(3\mu^2 - 1) + \frac{7}{2}\phi_3(5\mu^3 - 3\mu) + \frac{1}{8}(63\mu^5 - 70\mu^3 + 15\mu) \Bigg] d\mu = \int_0^1 \frac{1}{2}(5\mu^3 - 3\mu)\phi_b d\mu \,, \quad \text{(D.2)}$$

for $i = 5$:

$$\int_0^1 \frac{1}{8}(63\mu^5 - 70\mu^3 + 15\mu) \Bigg[\phi_0 + 3\phi_1\mu + \frac{5}{2}\phi_2(3\mu^2 - 1) + \frac{7}{2}\phi_3(5\mu^3 - 3\mu) + \frac{1}{8}(63\mu^5 - 70\mu^3 + 15\mu) \Bigg] d\mu = \int_0^1 \frac{1}{8}(63\mu^5 - 70\mu^3 + 15\mu)\phi_b d\mu \,, \quad \text{(D.3)}$$

and for $i = 7$:

$$\int_0^1 \frac{1}{16}(429\mu^7 - 693\mu^5 + 315\mu^3 + 35\mu)\Bigg[\phi_0 + 3\phi_1\mu + \frac{5}{2}\phi_2(3\mu^2 - 1) + \frac{7}{2}\phi_3(5\mu^3 - 3\mu) + \frac{1}{8}(63\mu^5 - 70\mu^3 + 15\mu) + \frac{1}{16}(429\mu^7 - 693\mu^5 + 315\mu^3 + 35\mu)\Bigg] d\mu = \int_0^1 \frac{1}{16}(429\mu^7 - 693\mu^5 + 315\mu^3 + 35\mu)\phi_b d\mu \,. \quad \text{(D.4)}$$

Carrying out the simple integrations yields the following system of equations:

$$\frac{1}{2}\phi_0 + \phi_1 + \frac{5}{8}\phi_2 - \frac{3}{16}\phi_4 + \frac{13}{128}\phi_6 = \frac{1}{2}\phi_b \quad \text{(D.5a)}$$

$$-\frac{1}{8}\phi_0 + \frac{5}{8}\phi_2 + \phi_3 + \frac{81}{128}\phi_4 - \frac{13}{64}\phi_6 = -\frac{1}{8}\phi_b \quad \text{(D.5b)}$$

$$\frac{1}{16}\phi_0 - \frac{25}{128}\phi_2 + \frac{81}{128}\phi_4 + \phi_5 + \frac{325}{512}\phi_6 = \frac{1}{16}\phi_b \quad \text{(D.5c)}$$

$$-\frac{5}{128}\phi_0 + \frac{7}{64}\phi_2 - \frac{105}{512}\phi_4 + \frac{325}{512}\phi_6 + \phi_7 = -\frac{5}{128}\phi_b \,, \quad \text{(D.5d)}$$

where ϕ_b is again an isotropic source prescribed on the planar boundary. For the odd moment in each boundary equation, we insert Eq (3.5) to remove them, leaving only the even moments and a set of first-order differential equations:

$$\frac{1}{2}\phi_0 + \frac{1}{3\Sigma_1}\frac{\partial}{\partial x}(\phi_0 + 2\phi_2) + \frac{5}{8}\phi_2 - \frac{3}{16}\phi_4 + \frac{13}{128}\phi_6 = \frac{1}{2}\phi_b \quad \text{(D.6a)}$$

$$-\frac{1}{8}\phi_0 + \frac{5}{8}\phi_2 + \frac{1}{7\Sigma_3}\frac{\partial}{\partial x}(3\phi_2 + 4\phi_4) + \frac{81}{128}\phi_4 - \frac{13}{64}\phi_6 = -\frac{1}{8}\phi_b \quad \text{(D.6b)}$$

$$\frac{1}{16}\phi_0 - \frac{25}{128}\phi_2 + \frac{81}{128}\phi_4 + \frac{1}{11\Sigma_5}\frac{\partial}{\partial x}(5\phi_4 + 6\phi_6) + \frac{325}{512}\phi_6 = \frac{1}{16}\phi_b \quad \text{(D.6c)}$$

$$-\frac{5}{128}\phi_0 + \frac{7}{64}\phi_2 - \frac{105}{512}\phi_4 + \frac{325}{512}\phi_6 + \frac{1}{15\Sigma_7}\frac{\partial}{\partial x}(7\phi_6) = -\frac{5}{128}\phi_b \,. \quad \text{(D.6d)}$$

We can again make the substitution of variables given by Eqs (3.9) for consistency with the equations defined on the domain. In addition, we apply the SP_N approximation to the derivatives by assuming they are instead multidimensional gradients on the

boundary such that $\frac{\partial}{\partial x} \rightarrow \hat{\mathbf{n}} \cdot \nabla$. Doing this gives:

$$\frac{1}{2}\Big(u_1 - \frac{2}{3}u_2 + \frac{8}{15}u_3 - \frac{16}{35}u_4\Big) + \frac{1}{3\Sigma_1}\hat{\mathbf{n}} \cdot \nabla\Big(\Big(u_1 - \frac{2}{3}u_2 + \frac{8}{15}u_3 - \frac{16}{35}u_4\Big) + 2\Big(\frac{1}{3}u_2 - \frac{4}{15}u_3 + \frac{8}{35}u_4\Big)\Big) + \frac{5}{8}\Big(\frac{1}{3}u_2 - \frac{4}{15}u_3 + \frac{8}{35}u_4\Big) - \frac{3}{16}\Big(\frac{1}{5}u_3 - \frac{6}{35}u_4\Big) + \frac{13}{128}\Big(\frac{1}{7}u_4\Big) = \frac{1}{2}\phi_b\,, \quad \text{(D.7)}$$

$$-\frac{1}{8}\Big(u_1 - \frac{2}{3}u_2 + \frac{8}{15}u_3 - \frac{16}{35}u_4\Big) + \frac{5}{8}\Big(\frac{1}{3}u_2 - \frac{4}{15}u_3 + \frac{8}{35}u_4\Big) + \frac{1}{7\Sigma_3}\hat{\mathbf{n}} \cdot \nabla\Big(3\Big(\frac{1}{3}u_2 - \frac{4}{15}u_3 + \frac{8}{35}u_4\Big) + 4\Big(\frac{1}{5}u_3 - \frac{6}{35}u_4\Big)\Big) + \frac{81}{128}\Big(\frac{1}{5}u_3 - \frac{6}{35}u_4\Big) - \frac{13}{64}\Big(\frac{1}{7}u_4\Big) = -\frac{1}{8}\phi_b\,, \quad \text{(D.8)}$$

$$\frac{1}{16}\Big(u_1 - \frac{2}{3}u_2 + \frac{8}{15}u_3 - \frac{16}{35}u_4\Big) - \frac{25}{128}\Big(\frac{1}{3}u_2 - \frac{4}{15}u_3 + \frac{8}{35}u_4\Big) + \frac{81}{128}\Big(\frac{1}{5}u_3 - \frac{6}{35}u_4\Big) + \frac{1}{11\Sigma_5}\hat{\mathbf{n}} \cdot \nabla\Big(5\Big(\frac{1}{5}u_3 - \frac{6}{35}u_4\Big) + 6\Big(\frac{1}{7}u_4\Big)\Big) + \frac{325}{512}\Big(\frac{1}{7}u_4\Big) = \frac{1}{16}\phi_b\,, \quad \text{(D.9)}$$

$$-\frac{5}{128}\Big(u_1 - \frac{2}{3}u_2 + \frac{8}{15}u_3 - \frac{16}{35}u_4\Big) + \frac{7}{64}\Big(\frac{1}{3}u_2 - \frac{4}{15}u_3 + \frac{8}{35}u_4\Big) - \frac{105}{512}\Big(\frac{1}{5}u_3 - \frac{6}{35}u_4\Big) + \frac{325}{512}\Big(\frac{1}{7}u_4\Big) + \frac{1}{15\Sigma_7}\hat{\mathbf{n}} \cdot \nabla\Big(7\Big(\frac{1}{7}u_4\Big)\Big) = -\frac{5}{128}\phi_b\,. \quad \text{(D.10)}$$

By rearranging the system such that we have a single gradient operator in each equation, we again have a matrix system:

$$\mathbf{n} \cdot D_n \nabla u_n + \sum_{m=1}^{4} B_{nm} u_m = s_n \qquad n = 1, 2, 3, 4\,, \quad \text{(D.11)}$$

where $\mathbf{D}$ and $\mathbf{u}$ are defined as before and:

$$\mathbf{s} = \Big(\frac{1}{2}\phi_b \;\; -\frac{1}{8}\phi_b \;\; \frac{1}{16}\phi_b \;\; -\frac{5}{128}\phi_b\Big)^T\,, \quad \text{(D.12)}$$

is the source vector on the boundary and

$$\mathbf{B} = \begin{bmatrix} \frac{1}{2} & -\frac{1}{8} & \frac{1}{16} & -\frac{5}{128} \\ -\frac{1}{8} & \frac{7}{24} & -\frac{41}{384} & \frac{1}{16} \\ \frac{1}{16} & -\frac{41}{384} & \frac{407}{1920} & -\frac{233}{2560} \\ -\frac{5}{128} & \frac{1}{16} & -\frac{233}{2560} & \frac{3023}{17920} \end{bmatrix}, \tag{D.13}$$

is a dense matrix of coefficients. Again, as pointed out by Evans, the SP_1 approximation reduces this boundary condition to the standard Marshak diffusion boundary condition. For reflecting boundary conditions, we use the same procedure as the P_N equations where the odd-moments are zero such that Eq (B.34) is true. From setting Eq (3.5) to zero for odd ϕ_n and again substituting Eq (3.9) we immediately find that:

$$\nabla \mathbf{u} = 0 \tag{D.14}$$

for reflecting SP_N boundaries, providing enough equations to close the system.

Analogously, for Marshak conditions on the boundaries of the multigroup problem we have:

$$\hat{\mathbf{n}} \cdot \mathbb{D}_n \nabla \mathbb{U}_n + \sum_{m=1}^{4} \mathbb{B}_{nm} \mathbb{U}_m = \mathbb{S}_n \,, \tag{D.15}$$

with $\mathbb{S}_n$ the vector of group-wise boundary source term on each boundary for each psuedo-moment and $\mathbb{B}_{nm}$ is an $N_g \times N_g$ diagonal matrix with the value B_{nm} on the diagonal. For reflecting conditions, again we have $\nabla \mathbb{U}_n = 0$ for all pseudo-moments.

Appendix E

Finite Volume Discretization for the SP_N Equations

The moment-discretized form of the SP_N equations given by Eq (3.11) for the monoenergetic case has yet to have the spatial components of the solution and the derivative operators discretized. To achieve this, we choose a conservation law approach, using Eq (3.11) as the balance equation, to yield a finite volume calculation (LeVeque, 2002). To do this, we first define a control volume about point (i, j, k) in the domain as a cell in a Cartesian mesh as shown in Figure E.1).

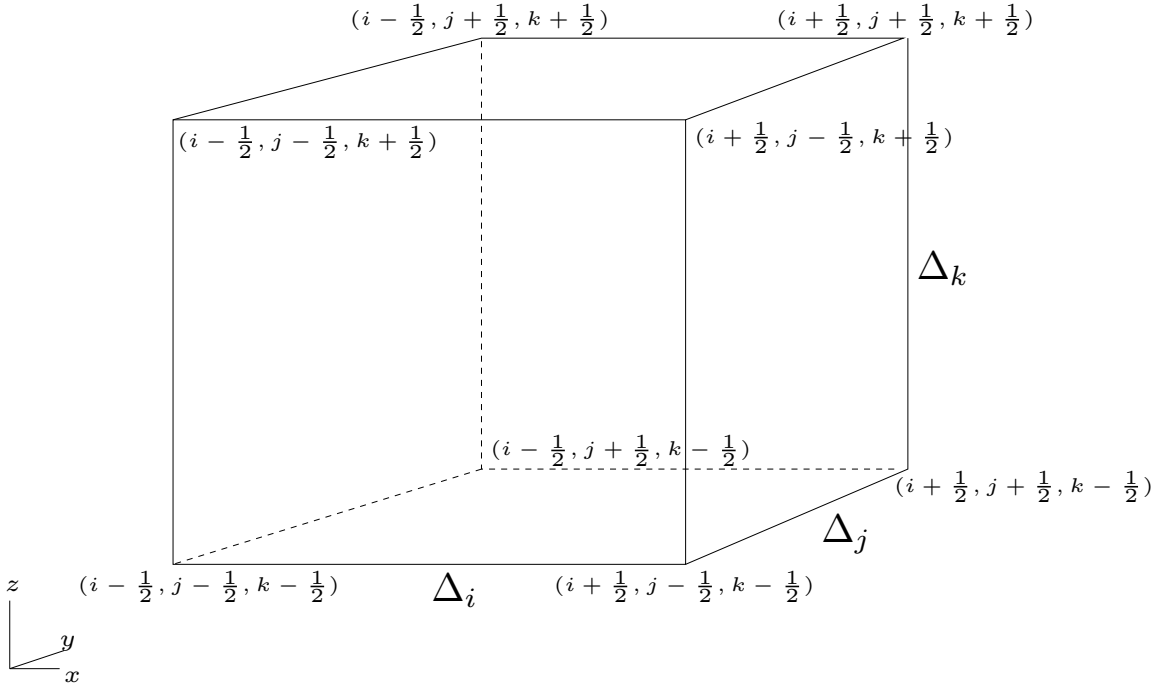

Figure E.1: **Cartesian mesh cell used for the spatial discretization of the SP_N equations.** *The cell is centered about (i, j, k) and has a volume $V_{ijk} = \Delta_i \Delta_j \Delta_k$.*

To begin, we first define the neutron current in the n^th pseudo-moment, u_n using Fick's law as:

$$J_n = -D_n \nabla u_n \,, \tag{E.1}$$

where J_n is the moment current and D_n is the moment diffusion coefficient. Next, we

substitue Eq (E.1) into Eq (3.11) to get the balance equation:

$$\nabla \cdot J_n + \sum_{m=1}^{4} A_{nm} u_m = q_n \qquad n = 1, 2, 3, 4 \,, \tag{E.2}$$

with the current term accounting for losses, q_n accounting for neutron generation, and $\sum_{m=1}^{4} A_{nm} u_m$ accounting for moment-to-moment transport. To enforce conservation, we integrate moments over the control volume of cell ijk in Figure E.1):

$$\int_{ijk} \left[\nabla \cdot J_n + \sum_{m=1}^{4} A_{nm} u_m - q_n \right] dV = 0 \qquad n = 1, 2, 3, 4 \,. \tag{E.3}$$

If for a given quantity X, we define the cell-averaged quantity X_{ijk} as:

$$X_{ijk} = \frac{\int_{ijk} X dV}{V_{ijk}} \,, \tag{E.4}$$

then Eq (E.3) can be written as:

$$\int_{ijk} \nabla \cdot J_n dV + \sum_{m=1}^{4} A_{nm,ijk} u_{m,ijk} = q_{n,ijk} \,, \tag{E.5}$$

with n now implied to be 1,2,3 or 4 and the moment matrix elements, moments, and sources now defined as volume averaged terms in the mesh cell. Using the divergence theorem, we can rewrite the remaining integral in terms of a discrete sum of area-weighted averages over the faces of the cell:

$$\int_{ijk} \nabla \cdot J_n dV = \int_{\partial} J_n \cdot \hat{n} dA = \sum_{f=1}^{6} J_{n,f} \cdot \hat{n}_f A_f \,, \tag{E.6}$$

where f is the face index, $\hat{n}_f$ is the unit normal vector for that face and A_f is the area of the face. Inserting this into Eq (E.5) and expanding the face current summation provides a discrete form of the SP_N equations:

$$\begin{aligned}(J_{n,i+1/2} - J_{n,i-1/2})\Delta_j \Delta_k + (J_{n,j+1/2} - J_{n,j-1/2})\Delta_i \Delta_k + \\ (J_{n,k+1/2} - J_{n,k-1/2})\Delta_i \Delta_j + \sum_{m=1}^{4} A_{nm,ijk} u_{m,ijk} V_{ijk} = q_{n,ijk} V_{ijk} \,,\end{aligned} \tag{E.7}$$

For the discretization we desire only cell-centered quanities and therefore we aim to elminate all $J_{n,l\pm1/2}$ terms from the discretization where $l = \{i, j, k\}$. To do this, we

enforce continuity of the moment flux and the moment currents across cell boundaries such that $J^+_{n,l\pm1/2} = J^-_{n,l\pm1/2}$. Considering $J_{n,l+1/2}$, using Eq (E.1) we then have:

$$J_{n,l+1/2} = -2D_{n,l+1/2}\frac{u_{n,l+1} - u_{n,l}}{\Delta_{l+1} + \Delta_l}\,, \tag{E.8}$$

$$J^+_{n,l+1/2} = -2D_{n,l+1}\frac{u_{n,l+1} - u_{n,l+1/2}}{\Delta_{l+1}}\,, \tag{E.9}$$

$$J^-_{n,l+1/2} = -2D_{n,l}\frac{u_{n,l+1/2} - u_{n,l}}{\Delta_l}\,. \tag{E.10}$$

To eliminate $D_{n,l+1/2}$ in Eq (E.8), we equate Eqs (E.9) and (E.10) to enforce continuity of the neutron current:

$$-2D_{n,l+1}\frac{u_{n,l+1} - u_{n,l+1/2}}{\Delta_{l+1}} = -2D_{n,l}\frac{u_{n,l+1/2} - u_{n,l}}{\Delta_l}\,. \tag{E.11}$$

Solving for $u_{n,l+1/2}$ gives:

$$u_{n,l+1/2} = \frac{\Delta_l D_{n,l+1}u_{n,l+1} + \Delta_{l+1}D_{n,l}u_{n,l}}{\Delta_l D_{n,l+1} + \Delta_{l+1}D_{n,l}}\,. \tag{E.12}$$

Next, we equate Eq (E.8) to Eq (E.9):

$$-2D_{n,l+1/2}\frac{u_{n,l+1} - u_{n,l}}{\Delta_{l+1} + \Delta_l} = -2D_{n,l+1}\frac{u_{n,l+1} - u_{n,l+1/2}}{\Delta_{l+1}}\,, \tag{E.13}$$

and rearrange:

$$\Delta_{l+1}D_{n,l+1/2}(u_{n,l+1} - u_{n,l}) = (\Delta_{l+1} + \Delta_l)D_{n,l+1}u_{n,l+1} - (\Delta_{l+1} + \Delta_l)D_{n,l+1}u_{n,l+1/2}\,. \tag{E.14}$$

Inserting Eq (E.12) into Eq (E.14) gives:

$$\begin{aligned}\Delta_{l+1}D_{n,l+1/2}(\Delta_l D_{n,l+1} + \Delta_{l+1}D_{n,l})(u_{n,l+1} - u_{n,l}) = \\ (\Delta_{l+1} + \Delta_l)D_{n,l+1}(\Delta_l D_{n,l+1} + \Delta_{l+1}D_{n,l})u_{n,l+1} - \\ (\Delta_{l+1} + \Delta_l)D_{n,l+1}(\Delta_l D_{n,l+1}u_{n,l+1} + \Delta_{l+1}D_{n,l}u_{n,l})\,.\end{aligned} \tag{E.15}$$

Expanding gives 5 terms:

$$\begin{aligned}\Delta_{l+1}D_{n,l+1/2}(\Delta_l D_{n,l+1}+\Delta_{l+1}D_{n,l})(u_{n,l+1}-u_{n,l}) = \\ (\Delta_{l+1}+\Delta_l)D_{n,l+1}\Delta_l D_{n,l+1}u_{n,l+1}+(\Delta_{l+1}+\Delta_l)D_{n,l+1}\Delta_{l+1}D_{n,l}u_{n,l+1}- \\ (\Delta_{l+1}+\Delta_l)D_{n,l+1}\Delta_l D_{n,l+1}u_{n,l+1}-(\Delta_{l+1}+\Delta_l)D_{n,l+1}\Delta_{l+1}D_{n,l}u_{n,l}\,, \quad \text{(E.16)}\end{aligned}$$

two of which cancel, giving:

$$\begin{aligned}\Delta_{l+1}D_{n,l+1/2}(\Delta_l D_{n,l+1}+\Delta_{l+1}D_{n,l})(u_{n,l+1}-u_{n,l}) = \\ (\Delta_{l+1}+\Delta_l)\Delta_{l+1}D_{n,l+1}D_{n,l}(u_{n,l+1}-u_{n,l})\,. \quad \text{(E.17)}\end{aligned}$$

Solving Eq (E.17) for $D_{n,l+1/2}$ then gives the face diffusion coefficient in terms of cell-centered coefficients:

$$D_{n,l+1/2}=\frac{\Delta_{l+1}D_{n,l+1}D_{n,l}+\Delta_l D_{n,l+1}D_{n,l}}{\Delta_l D_{n,l+1}+\Delta_{l+1}D_{n,l}}\,. \quad \text{(E.18)}$$

With this diffusion coefficient, we can then finish the derivation of the cell-face currents by inserting Eq (E.18) into Eq (E.8) giving:

$$J_{n,l+1/2}=-2\frac{D_{n,l+1}D_{n,l}}{\Delta_l D_{n,l+1}+\Delta_{l+1}D_{n,l}}(u_{n,l+1}-u_{n,l})\,. \quad \text{(E.19)}$$

Analagously, for cell face in the opposite direction, we have:

$$J_{n,l-1/2}=-2\frac{D_{n,l}D_{n,l-1}}{\Delta_{l-1}D_{n,l}+\Delta_l D_{n,l-1}}(u_{n,l}-u_{n,l-1})\,. \quad \text{(E.20)}$$

Before inserting the derived face fluxes into Eq (E.7), we divide by the cell volume to get:

$$\begin{aligned}(J_{n,i+1/2}-J_{n,i-1/2})\frac{1}{\Delta_i}+(J_{n,j+1/2}-J_{n,j-1/2})\frac{1}{\Delta_j}+ \\ (J_{n,k+1/2}-J_{n,k-1/2})\frac{1}{\Delta_k}+\sum_{m=1}^{4}A_{nm,ijk}u_{m,ijk}=q_{n,ijk}\,, \quad \text{(E.21)}\end{aligned}$$

Based on the face currents computed, we identify the following coefficients that will

appear in the moment equations:

$$C_{n,l+1} = \frac{2}{\Delta_l}\left(\frac{D_{n,l+1}D_{n,l}}{\Delta_l D_{n,l+1} + \Delta_{l+1} D_{n,l}}\right), \tag{E.22}$$

$$C_{n,l-1} = \frac{2}{\Delta_l}\left(\frac{D_{n,l}D_{n,l-1}}{\Delta_{l-1} D_{n,l} + \Delta_l D_{n,l-1}}\right). \tag{E.23}$$

Inserting these coefficients into Eqs (E.19) and (E.20) gives:

$$J_{n,l+1/2} = -\Delta_l C_{n_l+1}(u_{n,l+1} - u_{n,l}) . \tag{E.24}$$

$$J_{n,l-1/2} = -\Delta_l C_{n_l-1}(u_{n,l} - u_{n,l-1}) . \tag{E.25}$$

Applying Eqs (E.24) and (E.25) Eq (E.21) gives the final spatially discretized SP_N equations:

$$-C_{n,i+1}u_{n_i+1} - C_{n,i-1}u_{n_i-1} - C_{n,j+1}u_{n_j+1} - C_{n,j-1}u_{n_j-1} - C_{n,k+1}u_{n_k+1} - C_{n,k-1}u_{n_k-1} + \sum_{m=1}^{4}\Big[A_{nm,ijk} + (C_{n,i+1} + C_{n,j+1} + C_{n,k+1} + C_{n,i-1} + C_{n,j-1} + C_{n,k-1})\delta_{nm}\Big]u_{m,ijk} = q_{n,ijk} . \tag{E.26}$$

On the boundaries of the domain, special treatment of the face currents must be made as their are no adjacent cells with which to close the system. Starting with Eq (D.11), we again insert the neutron current given by Eq (E.1):

$$\mathbf{n} \cdot J_n + \sum_{m=1}^{4} B_{nm}u_m = s_n . \tag{E.27}$$

On the low boundary we then have:

$$J_{n,1/2} = s_{n,1} - \sum_{m=1}^{4} B_{nm,1/2}u_{m,1/2} , \tag{E.28}$$

or in terms of moments:

$$J_{n,1/2} = -2D_{n,1}\frac{u_{n,1} - u_{n,1/2}}{\Delta_1} . \tag{E.29}$$

Equating Eqs (E.28) and (E.29) gives an extra equation for the boundary moments:

$$s_{n,1} = \sum_{m=1}^{4} B_{nm,1/2} u_{m,1/2} - 2D_{n,1} \frac{u_{n,1} - u_{n,1/2}}{\Delta_1} \,, \tag{E.30}$$

or

$$s_{n,1} = \sum_{m=1}^{4} \left[B_{nm,1/2} + \frac{2D_{n,1}}{\Delta_1} \delta_{mn} \right] u_{m,1/2} - 2\frac{D_{n,1}}{\Delta_1} u_{n,1} \,. \tag{E.31}$$

To use these boundary fluxes to close the system, we insert Eq (E.29) into Eq (E.21) and then add Eq (E.31) to the system to eliminate the boundary currents introduced by Eq (E.29).

For the multigroup SP_N equations, the spatial discretization is identical, this time with the multigroup moment vectors where now:

$$-\mathbb{C}_{n,i+1}\mathbb{U}_{n_i+1} - \mathbb{C}_{n,i-1}\mathbb{U}_{n_i-1} - \mathbb{C}_{n,j+1}\mathbb{U}_{n_j+1} - \mathbb{C}_{n,j-1}\mathbb{U}_{n_j-1} - \mathbb{C}_{n,k+1}\mathbb{U}_{n_k+1} - \mathbb{C}_{n,k-1}\mathbb{U}_{n_k-1} + \sum_{m=1}^{4} \left[\mathbb{A}_{nm,ijk} + (\mathbb{C}_{n,i+1} + \mathbb{C}_{n,j+1} + \mathbb{C}_{n,k+1} + \mathbb{C}_{n,i-1} + \mathbb{C}_{n,j-1} + \mathbb{C}_{n,k-1})\delta_{nm} \right] \mathbb{U}_{m,ijk} = \mathbb{Q}_{n,ijk}, \tag{E.32}$$

and the coefficients are:

$$\mathbb{C}_{n,l+1} = \frac{2}{\Delta_l} \left(\frac{\mathbb{D}_{n,l+1}\mathbb{D}_{n,l}}{\Delta_l \mathbb{D}_{n,l+1} + \Delta_{l+1}\mathbb{D}_{n,l}} \right) , \tag{E.33}$$

$$\mathbb{C}_{n,l-1} = \frac{2}{\Delta_l} \left(\frac{\mathbb{D}_{n,l}\mathbb{D}_{n,l-1}}{\Delta_{l-1} \mathbb{D}_{n,l} + \Delta_l \mathbb{D}_{n,l-1}} \right) . \tag{E.34}$$

The boundary conditions are analagously formulated with the multigroup matrices/vectors.

Appendix F

Two-Dimensional One-Speed Neutron Diffusion Model Problem

For many numerical experiments in this work, we choose the one-speed, two-dimensional neutron diffusion equation as a model neutron transport problem (Duderstadt and Hamilton, 1976):

$$-\boldsymbol{\nabla}\cdot D\boldsymbol{\nabla}\phi+\Sigma_a\phi=S\,, \tag{F.1}$$

where ϕ is the neutron flux, Σ_a is the absorption cross section, and S is the source of neutrons. In addition, D is the diffusion coefficient defined as:

$$D=\frac{1}{3(\Sigma_t-\bar{\mu}\Sigma_s)}\,, \tag{F.2}$$

where Σ_s is the scattering cross section, $\Sigma_t=\Sigma_a+\Sigma_s$ is the total cross section, and $\bar{\mu}$ is the cosine of the average scattering angle. For simplicity, we will take $\bar{\mu}=0$ for our analysis giving $D=(3\Sigma_t)^{-1}$. In addition, to further simplify we will assume a homogeneous domain such that the cross sections remain constant throughout. Doing this permits us to rewrite Eq (F.1) as:

$$-D\boldsymbol{\nabla}^2\phi+\Sigma_a\phi=S\,. \tag{F.3}$$

We choose a finite difference scheme on a square Cartesian grid to discretize the problem. For the Laplacian, we choose the 9-point stencil shown in Figure F.1 over a grid of size h (LeVeque, 2007):

$$\begin{aligned}\nabla_9^2\phi=\frac{1}{6h^2}[&4\phi_{i-1,j}+4\phi_{i+1,j}+4\phi_{i,j-1}+4\phi_{i,j+1}+\phi_{i-1,j-1}\\&+\phi_{i-1,j+1}+\phi_{i+1,j-1}+\phi_{i+1,j+1}-20\phi_{i,j}]\,.\end{aligned} \tag{F.4}$$

We then have the following linear system to solve:

$$\begin{aligned}-\frac{1}{6h^2}[&4\phi_{i-1,j}+4\phi_{i+1,j}+4\phi_{i,j-1}+4\phi_{i,j+1}+\phi_{i-1,j-1}\\&+\phi_{i-1,j+1}+\phi_{i+1,j-1}+\phi_{i+1,j+1}-20\phi_{i,j}]+\Sigma_a\phi_{i,j}=s_{i,j}\,,\end{aligned} \tag{F.5}$$

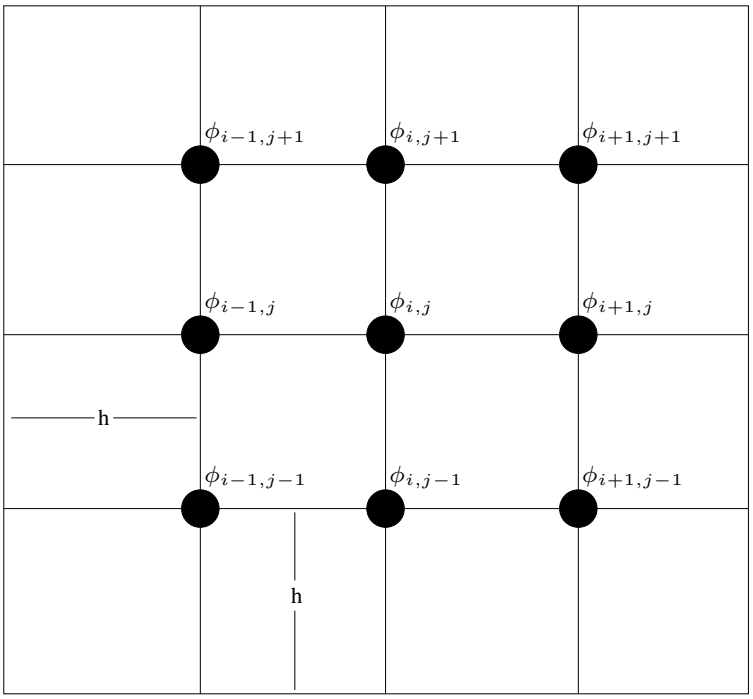

Figure F.1: **Nine-point Laplacian stencil.**

and in operator form:

$$\mathbf{D}\boldsymbol{\phi} = \mathbf{s}\,, \tag{F.6}$$

where $\mathbf{D}$ is the diffusion operator, $\mathbf{s}$ is the source in vector form and $\boldsymbol{\phi}$ is the vector of unknown fluxes.

To close the system, a set of boundary conditions is required. In the case of a non-reentrant current condition applied to all global boundaries of the domain, we choose the formulation of Duderstadt by assuming the flux is zero at some ghost point beyond the grid. Consider for example the equations on the $i = 0$ boundary of the domain:

$$-\frac{1}{6h^2}[4\phi_{-1,j} + 4\phi_{1,j} + 4\phi_{0,j-1} + 4\phi_{0,j+1} + \phi_{-1,j-1} + \phi_{-1,j+1} + \phi_{1,j-1} + \phi_{1,j+1} - 20\phi_{0,j}] + \Sigma_a\phi_{0,j} = s_{0,j}\,. \tag{F.7}$$

Here we note some terms where $i = -1$ and therefore are representative of grid points beyond the boundary of the domain. We set the flux at these points to be zero, giving

a valid set of equations for the $i = 0$ boundary:

$$-\frac{1}{6h^2}[4\phi_{1,j} + 4\phi_{0,j-1} + 4\phi_{0,j+1} + \phi_{-1,j+1} + \phi_{1,j-1} + \phi_{1,j+1} - 20\phi_{0,j}] + \Sigma_a\phi_{0,j} = s_{0,j} \ . \quad \text{(F.8)}$$

We repeat this procedure for the other boundaries of the domain. For reflecting boundary conditions, the net current across a boundary is zero.

Appendix G

Conventional Solution Methods for Nonlinear Systems

G.1 Inexact Newton Methods

Inexact Newton methods arise when the Jacobian operator is not exactly inverted, resulting in an inexact Newton correction as initially described by Dembo and others (Dembo et al., 1982). For common sparse nonlinear systems, which in turn yield a sparse Jacobian matrix, this situation occurs when conventional iterative methods are applied. In their definition, Dembo formulated inexact methods such that they are independent of the linear method used to solve for the Newton correction and therefore are amenable to use with any linear solver. Furthermore, they bind the convergence of the outer nonlinear iteration to the inner linear iteration such that:

$$||\mathbf{J}(\mathbf{u}^k)\delta\mathbf{u}^k + \mathbf{F}(\mathbf{u}^k)|| \leq \eta^k ||\mathbf{F}(\mathbf{u}^k)|| \,, \tag{G.1}$$

where $\eta^k \in [0, 1)$ is defined as the *forcing term* at the k iterate. Eq (G.1) then states that the residual generated by the linear solver is bound by the nonlinear residual and how tightly it is bound is defined by the forcing term. This is useful in that we can vary how tightly coupled the convergence of the linear iterations used to generate the Newton correction is to the nonlinear iteration by relaxing or tightening the convergence properties on the linear iterative method. As a result, strategies for determining the forcing term can vary depending on the problem type and can greatly affect the convergence of the method or even prohibit convergence (Eisenstat and Walker, 1996). In addition, *globalization methods* may be used to modify the Newton correction in a more desire able direction such that convergence properties can be improved when the initial guess for $\mathbf{u}$ is poor (Pawlowski et al., 2006).

G.2 Newton-Krylov Methods

A form of inexact Newton methods, *Newton-Krylov methods* are nonlinear iterative methods that leverage a Krylov subspace method as the linear solver for generating the

Newton correction (Kelley, 1995). As we investigated in Chapter A, Krylov methods are robust and enjoy efficient parallel implementations on modern architectures. Furthermore, their lack of explicit dependence on the operator make them easier to implement than other methods. Additionally, although many iterations can become memory intensive due to the need to store the Krylov subspace for the orthogonalization procedure, at each nonlinear iteration this cost is reset as the Jacobian matrix will change due to its dependence on the solution vector. This means that for every nonlinear iteration, a completely new linear system is formed for generating the Newton correction and we can modify the Krylov solver parameters accordingly to accommodate this. In most nonlinear problems, the Jacobian operator is generally non-symmetric and therefore either Krylov methods with long recurrence relations that can handle non-symmetric systems must be considered or the Newton correction system must be preconditioned such that the operator is symmetric and short recurrence relation methods can be potentially be used.

With many Krylov methods available, which to use with the Newton method is dependent on many factors including convergence rates and memory usage. Several studies have been performed to investigate this (McHugh and Knoll, 1993; Knoll and McHugh, 1995). In their numerical studies in 1995, Knoll and McHugh used the set of highly nonlinear and stiff convection-diffusion-reaction equations to solve a set of tokamak plasma problems with the goal of measuring solver performance with Newton's method. They note several trade offs in using Krylov methods with the Newton solver. The first is that the optimization condition that results from the constraints (e.g. the minimization of the GMRES residual over the Krylov space) can be relaxed by restricting the size of the subspace such that only a fixed number of subspace vectors may be maintained, thus reducing memory requirements. We can also relax the optimization condition by instead restarting the recurrence relation with a new set of vectors once a certain number of vectors have been generated. The optimization condition is maintained over that particular set of vectors, however, Knoll and McHugh note that this ultimately slows the convergence rate as compared to keeping all vectors as the new set of vectors is not necessarily orthogonal to the previous set, and therefore not optimal over the entire iteration procedure. The orthogonality condition can be relaxed by using a recurrence relation that does not generate a strictly orthonormal basis for the Krylov subspace such as the Lanzcos biorthogonalization procedure, resulting in memory savings due to the shorter Lanzcos recurrence relation.

As a comparison, Knoll and McHugh chose an Arnoldi-based GMRES with a

fixed vector basis approximately the size of the number of iterations required to converge as the long recurrence relation solver and conjugate gradients squared (CGS), bi-orthogonalized conjugate gradient stabilized (Bi-CGSTAB), and transpose-free quasiminimal residual (TFQMR) methods as Lanzcos-based short recurrence relation solvers. All solvers were used to compute the right-preconditioned Newton correction system. For standard implementations of Newton's method where the Jacobian operator was explicitly formed using difference equations, all methods exhibited roughly equivalent iteration count performance for both the inner linear iterations and the outer nonlinear iterations in terms of iterations required to converge. Bi-CGSTAB typically performed the best for implementations where the Jacobian was explicitly formed and GMRES performing best for matrix-free implementations. However, upon investigating the convergence of the inner iterations, it was observed that the GMRES solver was significantly more robust, always generating a monotonically decreasing residual as compared to the Lanzcos-based methods which had the tendency to oscillate. Based on these results, in all of their future work Knoll and McHugh tended to use GMRES as the Krylov solver (Knoll and Keyes, 2004).

G.2.1 Jacobian-Free Approximation

In most cases, the Jacobian is difficult to form from the difference equations and costly to evaluate for large equation sets. For simple nonlinear cases such as the Navier-Stokes equations, the derivatives can be computed and coded, but due to the complexity of those derivatives and the resulting difference equations this task can be tedious, error prone, and must be repeated for every equation set. Furthermore, in their 1995 work, Knoll and McHugh also noted that a dominating part of their computation time was the evaluation of the difference equations for building the Jacobian (Knoll and McHugh, 1995). By recognizing that Krylov methods only need the action of the operator on the vector instead of the operator itself, the Jacobian can instead be approximated through various numerical methods including a difference-based Jacobian-free formulation.

Jacobian-Free methods, and in particular *Jacobian-Free Newton-Krylov* (JFNK) methods (Knoll and Keyes, 2004), rely on forming the action of the Jacobian on a vector as required by the Krylov solver through a forward difference scheme. In this case, the action of the Jacobian on some vector $\mathbf{v}$ is given as:

$$\mathbf{J}(\mathbf{u})\mathbf{v} = \frac{\mathbf{F}(\mathbf{u} + \epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})}{\epsilon}\,, \tag{G.2}$$

where ϵ is a small number typically on the order of machine precision. Kelley (Kelley, 1995) points out a potential downfall of this formulation in that if the discretization error in $\mathbf{F}(\mathbf{u})$ is on the order of the perturbation parameter ϵ, then the finite difference error from Eq (G.2) pollutes the solution. In addition, Knoll and McHugh noted that for preconditioning purposes, part of the Jacobian must still explicitly be formed periodically and that linear solver robustness issues were magnified by the matrix-free approach due to the first-order approximation. This formation frequency coupled with the numerous evaluations of the Jacobian approximation create a situation where after so many nonlinear iterations, it becomes cheaper to instead fully form the Jacobian. For simple equation sets, this may only take 5-10 Newton iterations to reach this point while over 30 may be required for larger equations sets and therefore larger Jacobians.

G.2.2 Automatic Differentiation for Jacobian Generation

If it is acceptable to store the actual Jacobian matrix, other methods are available to construct it without requiring hand-coding and evaluating derivatives, thus eliminating the associated issues. In addition, if any additional equations are added to the system or a higher order functional approximation is desired, it would be useful to avoid regenerating and coding these derivatives. Becoming more prominent in the 1990's, *automatic differentiation* is a mechanism by which the derivatives of a function can be generated automatically by evaluating it. Automatic differentiation is built on the concept that all functions discretely represented in a computer are ultimately represented by elementary mathematical operations. If the chain rule is applied to those elementary operations, then the derivatives of those functions can be computed to the order of accuracy of their original discretization in a completely automated way (Averick et al., 1994).

The work of Bartlett and others (Bartlett et al., 2006) extended initial Fortran-based work in the area of automatic differentiation implementations to leverage the parametric type and operator overloading features of C++ (Stroustrup, 1997). They formulate the differentiation problem from an element viewpoint by assuming that a global Jacobian can be assembled from local element function evaluations of $e_k : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{m_k}$, similar to the finite element assembly procedure as:

$$\mathbf{J}(\mathbf{u}) = \sum_{i=1}^{N} \mathbf{Q}_i^T \mathbf{J}_k \mathbf{P}_i \,, \tag{G.3}$$

where $\mathbf{J}_{k_i} = \partial e_{k_i} / \partial P_i u$ is the k^{th} element function Jacobian, $\mathbf{Q} \in \mathbb{R}^{n_{k_i} \times N}$ is a projector

onto the element domain and $\mathbf{P} \in \mathbb{R}^{m_{k_i} \times N}$ a projector onto the element range for $\mathbf{F}(\mathbf{u}) \in \mathbb{R}^{N \times N}$. The Jacobian matrix for each element will therefore have entirely local data in a dense structure, eliminating the need for parallel communication and sparse techniques during differentiation. Only when all local differentials are computed does communication of the Jacobian occur through gather/scatter operations in order to properly assembly it. Also of benefit is the fact that element-level computations generally consist of a smaller number of degrees of freedom, thus reducing memory requirements during evaluation as compared to a global formulation of the problem. Such a formulation is not limited to finite element formulations and is amenable to any scheme where the system is globally sparse with degrees of freedom coupled to local domains including finite volume representations. The templating capabilities of C++ were leveraged with the element-based evaluation and assembly scheme as in Eq (G.3) by templating element function evaluation code on the evaluation type. If these functions are instantiated with standard floating point types then the residual is returned. If they are instead instantiated with the operator-overloaded automatic differentiation types, both the residual and Jacobian are returned.

Of interest to Bartlett, Averick, and the many others that have researched automatic differentiation are measures of its performance relative to hand-coded derivatives and capturing the Jacobian matrix from matrix-free approximations. Given their element-based function evaluation scheme, Bartlett's work varied the number of degrees of freedom per element and compared both the floating point operation count and CPU time for both the templated automatic differentiation method and hand-coded derivatives for Jacobian evaluations. Although they observed a 50% increase in floating point operations in the templated method over the hand-coded method, run times were observed to be over 3 times faster for the templated method. They hypothesize that this is due to the fact that the element-based formulation of the templated method is causing better utilization of cache and therefore faster data access. Furthermore, they observed linear scaling behavior for automatic differentiation as the number of degrees of freedom per element were increased up to a few hundred. Based on these results, this type of automatic differentiation formulation was deemed acceptable for use in large-scale, production physics codes.

Appendix H

Parallel Scaling Study Data

I will put the raw data from the scaling studies here

references

Alexandrov, V., E. Atanassov, and I. Dimov. 2004. Parallel quasi-monte carlo methods for linear algebra problems. *Monte Carlo Methods & Applications* 10(3/4): 213–219.

Alexandrov, V.N. 1998. Efficient parallel monte carlo methods for matrix computations. *Mathematics and Computers in Simulation* 47(2âŁ“5):113–122.

Averick, Brett M., Jorge J. More, Christian H. Bischof, Alan Carle, and Andreas Griewank. 1994. Computing large sparse jacobian matrices using automatic differentiation. *SIAM Journal on Scientific Computing* 15(2):285–294.

Azmy, Yousry, and Enrico Sartori. 2010. *Nuclear computational science: A century in review*. 1st ed. Springer.

Baker, C. G., U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist. 2009. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Trans. Math. Softw.* 36(3):1–23.

Bartlett, Roscoe A., David M. Gay, and Eric T. Phipps. 2006. Automatic differentiation of c++ codes for large-scale scientific computing. In *Computational science - ICCS 2006*, vol. 3994, 525–532. Berlin, Heidelberg: Springer Berlin Heidelberg.

Branford, S., C. Sahin, A. Thandavan, C. Weihrauch, V.N. Alexandrov, and I.T. Dimov. 2008. Monte carlo methods for matrix computations on the grid. *Future Generation Computer Systems* 24(6):605–612.

Brantley, Patrick S., and Edward Larsen. 2000. The simplified p3 approximation. *Nuclear Science and Engineering* 134(1):1–21.

Brunner, Thomas A., and Patrick S. Brantley. 2009. An efficient, robust, domain-decomposition algorithm for particle monte carlo. *Journal of Computational Physics* 228(10):3882–3890.

Brunner, Thomas A., Todd J. Urbatsch, Thomas M. Evans, and Nicholas A. Gentile. 2006. Comparison of four parallel algorithms for domain decomposed implicit monte carlo. *Journal of Computational Physics* 212(2):527–539.

Cai, Xiao-Chuan, and David E. Keyes. 2002. Nonlinearly preconditioned inexact newton algorithms. *SIAM Journal on Scientific Computing* 24(1):183–200.

Danilov, D.L., S.M. Ermakov, and J.H. Halton. 2000. Asymptotic complexity of monte carlo methods for solving linear systems. *Journal of Statistical Planning and Inference* 85(1-2):5–18.

De Vahl Davis, G. 1983. Natural convection of air in a square cavity: A bench mark numerical solution. *International Journal for Numerical Methods in Fluids* 3(3): 249–264.

Dembo, Ron S., Stanley C. Eisenstat, and Trond Steihaug. 1982. Inexact newton methods. *SIAM Journal on Numerical Analysis* 19(2):400–408.

Devine, K., E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. 2002. Zoltan data management services for parallel dynamic applications. *Computing in Science Engineering* 4(2):90 –96.

Dimov, I., V. Alexandrov, and A. Karaivanova. 2001. Parallel resolvent monte carlo algorithms for linear algebra problems. *Mathematics and Computers in Simulation* 55(1âŁ“3):25–35.

Dimov, I.T., and V.N. Alexandrov. 1998. A new highly convergent monte carlo method for matrix computations. *Mathematics and Computers in Simulation* 47(2âŁ“5): 165–181.

Dimov, I.T., T.T. Dimov, and T.V. Gurov. 1998. A new iterative monte carlo approach for inverse matrix problem. *Journal of Computational and Applied Mathematics* 92(1):15–35.

Dimov, I.T., B. Philippe, A. Karaivanova, and C. Weihrauch. 2008. Robustness and applicability of markov chain monte carlo algorithms for eigenvalue problems. *Applied Mathematical Modelling* 32(8):1511–1529.

Dimov, Ivan, Vassil Alexandrov, Rumyana Papancheva, and Christian Weihrauch. 2007. Monte carlo numerical treatment of large linear algebra problems. In *Computational science âŁ“ ICCS 2007*, ed. Yong Shi, Geert Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, 747–754. Lecture Notes in Computer Science 4487, Springer Berlin Heidelberg.

Duderstadt, James J., and Louis J. Hamilton. 1976. *Nuclear reactor analysis*. 1st ed. Wiley.

Eisenstat, Stanley C., and Homer F. Walker. 1996. Choosing the forcing terms in an inexact newton method. *SIAM Journal on Scientific Computing* 17(1):16–32.

Evans, Katherine J., D.A. Knoll, and Michael Pernice. 2006. Development of a 2-d algorithm to simulate convection and phase transition efficiently. *Journal of Computational Physics* 219(1):404–417.

———. 2007. Enhanced algorithm efficiency for phase change convection using a multigrid preconditioner with a SIMPLE smoother. *Journal of Computational Physics* 223(1):121–126.

Evans, T. M. 2013. Simplified PN methods in denovo. Technical Report RNSD-00-000, Oak Ridge National Laboratory.

Evans, Thomas, and Scott Mosher. 2009. A monte carlo synthetic acceleration method for the non-linear, time-dependent diffusion equation. *American Nuclear Society - International Conference on Mathematics, Computational Methods and Reactor Physics 2009*.

Evans, Thomas, Scott Mosher, and Stuart Slattery. 2012. A monte carlo synthetic-acceleration method for solving the thermal radiation diffusion equation. *Journal of Computational Physics* Submitted.

Evans, Thomas, Alissa Stafford, Rachel Slaybaugh, and Kevin Clarno. 2010. Denovo: A new three-dimensional parallel discrete ordinates code in SCALE. *Nuclear Technology* 171(2):171–200.

Evans, Thomas, Todd Urbatsch, H Lichtenstein, and Morel. 2003. A residual monte carlo method for discrete thermal radiative diffusion. *Journal of Computational Physics* 189(2):539–556.

Forsythe, George E., and Richard A. Leibler. 1950. Matrix inversion by a monte carlo method. *Mathematical Tables and Other Aids to Computation* 4(31):127–129. ArticleType: research-article / Full publication date: Jul., 1950 / Copyright 1950 American Mathematical Society.

Gartling, David K. 1990. A test problem for outflow boundary conditions - flow over a backward-facing step. *International Journal for Numerical Methods in Fluids* 11(7):953 – 967.

Gaston, D, G Hansen, S Kadioglu, D A Knoll, C Newman, H Park, C Permann, and W Taitano. 2009. Parallel multiphysics algorithms and software for computational nuclear engineering. *Journal of Physics: Conference Series* 180:012012.

Gentile, N.A., Malvin Kalos, and Thomas A. Brunner. 2005. Obtaining identical results on varying numbers of processors in domain decomposed particle monte carlo simulations. In *Computational methods in transport*, vol. 48, 423–433. Berlin/Heidelberg: Springer-Verlag.

Ghia, U, K.N Ghia, and C.T Shin. 1982. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of Computational Physics* 48(3):387–411.

Godoy, William F., and Xu Liu. 2012. Parallel jacobian-free newton krylov solution of the discrete ordinates method with flux limiters for 3D radiative transfer. *Journal of Computational Physics* 231(11):4257–4278.

Gropp, W., and B. Smith. 1993. Scalable, extensible, and portable numerical libraries. In *Scalable parallel libraries conference, 1993., proceedings of the*, 87 –93.

Gropp, William D, Dinesh K Kaushik, David E Keyes, and Barry F Smith. 2001. High-performance parallel implicit CFD. *Parallel computing in aerospace* 27(4): 337–362.

Halton, J. H. 1962. Sequential monte carlo. *Mathematical Proceedings of the Cambridge Philosophical Society* 58(01):57–78.

———. 1994. Sequential monte carlo techniques for the the solution of linear systems. *Journal of Scientific Computing* 9:213–257.

Halton, John H. 1970. A retrospective and prospective survey of the monte carlo method. *SIAM Review* 12(1):1–63.

———. 2006. Sequential monte carlo techniques for solving non-linear systems. *Monte Carlo Methods & Applications* 12(2):113–141.

Hammersley, John Michael, and David Christopher Handscomb. 1964. *Monte carlo methods*. Methuen.

Heroux, Michael A., Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P.

Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. 2005. An overview of the trilinos project. *ACM Trans. Math. Softw.* 31(3):397–423.

Ji, Hao, and Yaohang Li. 2012. Reusing random walks in monte carlo methods for linear systems. *Procedia Computer Science* 9(0):383–392.

Kelley, C. T. 1995. *Iterative methods for linear and nonlinear equations*. 1st ed. Society for Industrial and Applied Mathematics.

Keyes, D. E. 1999. *How scalable is domain decomposition in practice?*

Keyes, David E., Dinesh K. Kaushik, and Barry F. Smith. 1997. Prospects for CFD on petaflops systems. Tech. Rep.

Knoll, D.A., and D.E. Keyes. 2004. Jacobian-free newton-krylov methods: a survey of approaches and applications. *Journal of Computational Physics* 193(2):357–397.

Knoll, D.A., and P.R. McHugh. 1995. Newton-krylov methods applied to a system of convection-diffusion-reaction equations. *Computer Physics Communications* 88(2-3): 141–160.

Kogge, Peter M., and Timothy J. Dysart. 2011. Using the TOP500 to trace and project technology and architecture trends. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*, 28:1–28:11. SC '11, New York, NY, USA: ACM.

LeVeque, Randall. 2007. *Finite difference methods for ordinary and partial differential equations: Steady-state and time-dependent problems*. SIAM, Society for Industrial and Applied Mathematics.

LeVeque, Randall J. 2002. *Finite volume methods for hyperbolic problems*. 1st ed. Cambridge University Press.

Lewis, E. E. 1993. *Computational methods of neutron transport*. Wiley-Interscience.

Li, Yaohang, and Michael Mascagni. 2003. Analysis of large-scale grid-based monte carlo applications. *The International Journal of High Performance Computing Applications* 17(4):369–382.

McHugh, P. R., and D. A. Knoll. 1992. *Fully implicit solutions of the benchmark backward facing step problem using finite element discretization and inexact newton's method*.

McHugh, Paul R., and Dana A. Knoll. 1993. Inexact newton's method solutions to the incompressible navier-stokes and energy equations using standard and matrix-free implementations. In *AIAA 11th computational fluid dynamics conference*, vol. -1, 385–393.

———. 1994. Fully coupled finite volume solutions of the incompressible NavierâŁ“Stokes and energy equations using an inexact newton method. *International Journal for Numerical Methods in Fluids* 19(5):439–455.

Mervin, Brenden, S.W. Mosher, Thomas Evans, John Wagner, and GI Maldonado. 2012. Variance estimation in domain decomposed monte carlo eigenvalue calculations. *PHYSOR 2012 - Advances in Reactor Physics*.

Musser, David R., and Alexander A. Stepanov. 1994. Algorithm-oriented generic libraries. *Software: Practice and Experience* 24(7):623–642.

Nachtigal, Noel M., Satish C. Reddy, and Lloyd N. Trefethen. 1992. How fast are nonsymmetric matrix iterations? *SIAM Journal on Matrix Analysis and Applications* 13(3):778–795.

Notz, P.K., and Pawlowski. 2010. Graph-based software design for managing complexity and enabling concurrency in multiphysics PDE software. *ACM Trans. Math. Softw.* 40:1–24.

Okten, Giray. 2005. Solving linear equations by monte carlo simulation. *SIAM Journal on Scientific Computing* 27(2):511–531.

Pawlowski, John N. Shadid, Thomas Smith, Eric Cyr, and Paula Weber. 2012. Drekar CFD - a turbulent fluid-flow and conjugate heat transfer code: Theory manual (version 1.0). Technical Report, Sandia National Laboratories.

Pawlowski, Roger P., John N. Shadid, Joseph P. Simonis, and Homer F. Walker. 2006. Globalization techniques for newton-krylov methods and applications to the fully coupled solution of the navier-stokes equations. *SIAM Review* 48(4):700–721.

Pernice, Michael, and Homer F. Walker. 1998. NITSOL: a newton iterative solver for nonlinear systems. *SIAM Journal on Scientific Computing* 19(1):302–318.

Pletcher, Richard H., John C. Tannehill, and Dale Anderson. 1997. *Computational fluid mechanics and heat transfer, second edition*. 2nd ed. Taylor & Francis.

Procassini, Richard, M.H. O'Brien, and J Taylor. 2005. Dynamic load balancing of parallel monte carlo transport calculations. *Monte Carlo 2005*.

Rief, H. 1999. Touching on a zero-variance scheme in solving linear equations by random walk processes. *Monte Carlo Methods & Applications* 5(2):135.

Romano, P., B. Forget, and F. Brown. 2010. Towards scalable parallelism in monte carlo particle transport codes using remote memory access. In *Joint international conference on supercomputing in nuclear applications and monte carlo 2010 (SNA+ MC2010)*, 17–21.

Saad, Youcef. 1993. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing* 14(2):9.

Saad, Youcef, and Martin H. Schultz. 1986. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 7(3):856–869.

Saad, Yousef. 2003. *Iterative methods for sparse linear systems*. SIAM.

Sabelfeld, K., and N. Mozartova. 2009. Sparsified randomization algorithms for large systems of linear equations and a new version of the random walk on boundary method. *Monte Carlo Methods & Applications* 15(3):257–284.

Shadid, John N., and Ray S. Tuminaro. 1994. A comparison of preconditioned nonsymmetric krylov methods on a large-scale MIMD machine. *SIAM Journal on Scientific Computing* 15(2):440–459.

Shadid, John N., Ray S. Tuminaro, and Homer F. Walker. 1997. An inexact newton method for fully coupled solution of the navier-stokes equations with heat and mass transport. *Journal of Computational Physics* 137(1):155–185.

Siegel, A., K. Smith, P. Fischer, and V. Mahadevan. 2012a. Analysis of communication costs for domain decomposed monte carlo methods in nuclear reactor analysis. *Journal of Computational Physics* 231(8):3119–3125.

Siegel, A.R., K. Smith, P.K. Romano, B. Forget, and K. Felker. 2012b. The effect of load imbalances on the performance of monte carlo algorithms in LWR analysis. *Journal of Computational Physics* (0).

Slaybaugh, Rachel N. 2011. Acceleration methods for massively parallel deterministic transport. PhD nuclear engineering and engineering physics, University of Wisconsin-Madison, Madison, WI.

Sosonkina, M., D.C.S. Allison, and L.T. Watson. 1998. Scalable parallel implementations of the GMRES algorithm via householder reflections. In *1998 international conference on parallel processing, 1998. proceedings*, 396 –404.

Spanier, Jerome, and Ely M. Gelbard. 1969. *Monte carlo principles and neutron transport problems*. New York: Dover Publications.

Srinivasan, A. 2010. Monte carlo linear solvers with non-diagonal splitting. *Mathematics and Computers in Simulation* 80(6):1133–1143.

Stroustrup, Bjarne. 1997. *The c++ programming language*. 3rd ed. Addison-Wesley Professional.

Trefethen, Lloyd N., and David Bau III. 1997. *Numerical linear algebra*. SIAM: Society for Industrial and Applied Mathematics.

Tuminaro, Ray S., John N. Shadid, and Scott A. Hutchinson. 1998. Parallel sparse matrix vector multiply software for matrices with data locality. *Concurrency: Practice and Experience* 10(3):229–247.

U.S. Department of Energy. 2011. CASL - a project summary. CASL-U-2011-0025-000, Consortium for Advanced Simulation of LWRs.

———. 2012. Resilient extreme-scale solvers. Tech. Rep. LAB 12-742, ASCR.

Vajargah, Behrouz Fathi. 2007. New advantages to obtain accurate matrix inversion. *Applied Mathematics and Computation* 189(2):1798–1804.

———. 2008. Stochastic inverse matrix computation with minimum variance of errors. *Applied Mathematics and Computation* 199(1):181–185.

Vajargah, Behrouz Fathi, and Kianoush Fathi Vajargah. 2006. Parallel monte carlo computations for solving SLAE with minimum communications. *Applied Mathematics and Computation* 183(1):1–9.

Wagner, JC, Scott Mosher, Thomas Evans, Doug Peplow, and John Turner. 2010. Hybrid and parallel domain-decomposition methods development to enable monte carlo for reactor analyses. *Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo*.

Wang, Qiqi, David Gleich, Amin Saberi, Nasrollah Etemadi, and Parviz Moin. 2008. A monte carlo method for solving unsteady adjoint equations. *Journal of Computational Physics* 227(12):6184–6205.

Wasow, W.R. 1952. A note on the inversion of matrices by random walks. *Mathematical Tables and Other Aids to Computation* 6(38):78–81.

Zienkiewicz, O. C., R. L. Taylor, and J. Z. Zhu. 2005. *The finite element method: Its basis and fundamentals, sixth edition*. 6th ed. Butterworth-Heinemann.