MASSIVELY PARALLEL MONTE CARLO METHODS FOR DISCRETE LINEAR AND NONLINEAR SYSTEMS

by

Stuart R. Slattery

A preliminary report 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

20 March 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

List of Figures

MASSIVELY PARALLEL MONTE CARLO METHODS FOR DISCRETE LINEAR AND NONLINEAR SYSTEMS

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 many fields of engineering and physics, linear and nonlinear 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 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 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 $\mathsf{O}(1,000,000)$ 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 preliminary report proposes the development of 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 motivation for advancing Monte Carlo solvers by providing multiphysics 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.

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

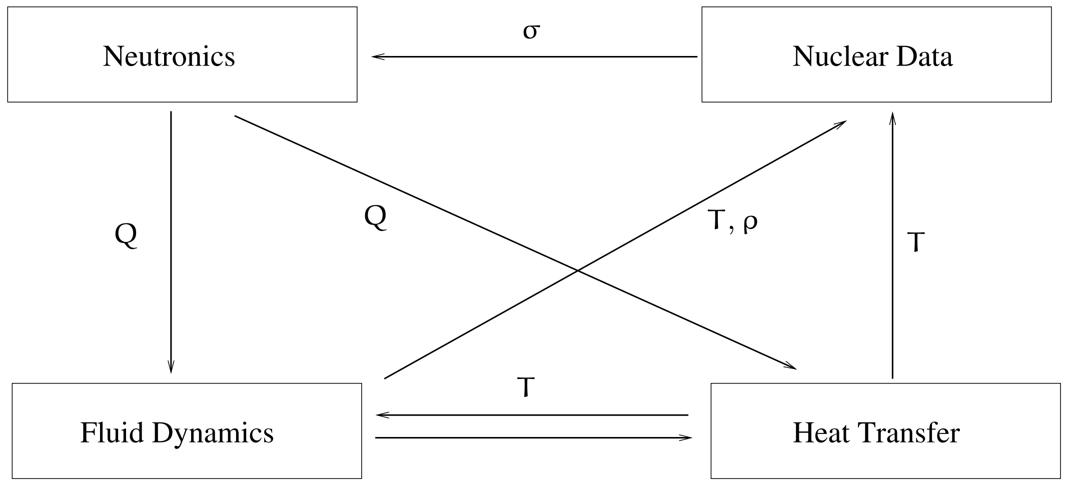

Figure 1.1: **Multiphysics dependency analysis of departure from nucleate boiling.** *A neutronics solution is required to compute power generation in the fuel pins, fluid dynamics is required to characterize boiling and fluid temperature and density, heat transfer is required to compute the fuel and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

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

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

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 $\mathsf{O}(1 \times 10^9)$ elements leveraging $\mathsf{O}(100,000)$ cores on leadership class machines. Neutronics components to be implemented in CASL for multiphysics analysis, such as the Denovo radiation transport code developed at Oak Ridge National Laboratory (Evans et al., 2010), compute trillions of unknowns for full core reactor analysis on $\mathsf{O}(1 \times 10^9)$ element meshes and $\mathsf{O}(100,000)$ 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.

1.2 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 will be 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 thesis proposes a new and novel method for nonlinear systems based on the MCSA method.

The Forward-Automated Newton-MCSA (FANM) method is proposed 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.

This preliminary report outlines in Chapter 2 the conventional methods used in practice for solving linear problems to provide a mathematical basis upon which to build new algorithms that aim to solve some of the aforementioned issues. Parallel schemes for conventional methods are also provided for background and understanding of how current methods may or may not map to future hardware. In addition, some components of their parallel implementations may be applied to stochastic methods development. In Chapter 3, Monte Carlo algorithms for solving linear systems and new parallel strategies will be outlined in full with links made to past work and their potential for offering improvements to the multiphysics analysis community. From these stochastic methods, a new nonlinear method is proposed in Chapter 4 and compared to conventional methods for solving nonlinear problems. The research proposal for this work is presented in Chapter 5 with progress to date reported including the development of a new finite element-based physics framework that leverages a preliminary general implementation of the stochastic method that will serve as a test bed for these new methods. An outline of a set of numerical experiments will then be provided that continues the development and implementation of these new methods and verifies them with benchmark solutions, culminating in the analysis of a multiphysics simulation of a pressurized water reactor subsystem using a production code system.

Chapter 2

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]

2.1 Preliminaries

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

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

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

where $\mathbf{A} \in \mathbb{R}^{\mathsf{N}\times\mathsf{N}}$ is a matrix operator such that $\mathbf{A} : \mathbb{R}^{\mathsf{N}} \rightarrow \mathbb{R}^{\mathsf{N}}$, $\mathbf{x} \in \mathbb{R}^{\mathsf{N}}$ is the solution vector, and $\mathbf{b} \in \mathbb{R}^{\mathsf{N}}$ 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:

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

In addition, for problems that have a large scale over which the independent variables may exist (e.g. a problem with events on timescales ranging from nanoseconds to hours), a good measure of this range is supplied by the *stiffness ratio*:

$$\mathsf{StiffnessRatio} = \frac{\max_{\lambda \in \sigma(\mathbf{A})} |\lambda|}{\min_{\lambda \in \sigma(\mathbf{A})} |\lambda|} \tag{2.7}$$

Those problems that have a wide range of scales in their independent variables, which will then be reflected in the operator, will then have a large stiffness ratio. We will define such problems with large stiffness ratios as *stiff*.

General to both matrices and vectors, *norms* are a mechanism for collapsing objects of many elements to a single value. Per LeVeque's text (LeVeque, 2007), the q-norm of a vector is defined as:

$$||\mathbf{v}||_q = \left[\sum_{i=1}^{N} |v_i|^q \right]^{1/q} , \ \mathbf{v} \in \mathbb{R}^N \ , \ q \in \mathbb{Z}^+ \tag{2.8}$$

where v_i is the i^{th} component of the vector. Depending on the value chosen for q, local or global qualities of the vector may be obtained. For example, $q = 2$ provides the root of a quadrature sum of all elements in the vector giving a global measure of the vector while $q = \infty$ gives the maximum value in the vector, a local quantity that does not give information regarding the other elements in the vector.

We can also compute the norm of a matrix by inferring from the norm of the vector on which it is operating. Per LeVeque, we search for a constant that is equivalent to $||\mathbf{A}||$:

$$||\mathbf{A}\mathbf{x}|| \leqslant C||\mathbf{x}|| \,, \tag{2.9}$$

where the minimum value of C that satisfies Eq (2.9) is equivalent to $||\mathbf{A}||$ and is valid $\forall \mathbf{x} \in \mathbb{R}^N$. The general definition in Eq (2.9) can be expanded in simple terms for common norms including the infinity norm:

$$||\mathbf{A}||_\infty = \max_{1 \leqslant i \leqslant N} \sum_{j=1}^{N} |a_{ij}| \,, \tag{2.10}$$

and the 2-norm:

$$||\mathbf{A}||_2 = \sqrt{\rho(\mathbf{A}^T\mathbf{A})} \,, \tag{2.11}$$

where ρ is the spectral radius as defined in Eq (2.6).

Knowing this, we can then define several useful properties of matrices including the *condition number*:

$$\kappa(\mathbf{A}) = ||\mathbf{A}|| \; ||\mathbf{A}^{-1}|| \,, \tag{2.12}$$

which gives as a metric on assessing how close to singular the system is. This is due to the fact $||\mathbf{A}^{-1}||$ is large near singularities (and undefined for a singular matrix) and thus a large condition number will be generated. We define such matrices as *ill-conditioned.*

2.2 Stationary Methods

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

$$\mathbf{A} = \mathbf{M} - \mathbf{N} \,, \tag{2.13}$$

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:

$$\mathbf{Mx} - \mathbf{Nx} = \mathbf{b} \,. \tag{2.14}$$

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

$$\mathbf{x} = \mathbf{H}\mathbf{x} + \mathbf{c}\,, \tag{2.15}$$

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:

$$\mathbf{x}^{k+1} = \mathbf{H}\mathbf{x}^{k} + \mathbf{c}\,, \tag{2.16}$$

with $k \in \mathbb{Z}^{+}$ defined as the *iteration index*. In general, we will define methods in the form of Eq (2.16) 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 (2.15) from Eq (2.16) to arrive at an error form of the linear problem:

$$\mathbf{e}^{k+1} = \mathbf{H}\mathbf{e}^{k}\,. \tag{2.17}$$

Our error after k iterations is then:

$$\mathbf{e}^{k} = \mathbf{H}^{k}\mathbf{e}^{0}\,. \tag{2.18}$$

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:

$$\mathbf{e}^{k} = \mathbf{R}\boldsymbol{\Lambda}^{k}\mathbf{R}^{-1}\mathbf{e}^{0}\,, \tag{2.19}$$

where $\boldsymbol{\Lambda}$ 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

[2]We may generalize this to non-diagonalizable matrices with the Jordan canonical form of $\mathbf{H}$.

then gives:

$$||\mathbf{e}^k||_2 \leqslant ||\mathbf{\Lambda}^k||_2 \; ||\mathbf{R}||_2 \; ||\mathbf{R}^{-1}||_2 \; ||\mathbf{e}^0||_2 \,, \tag{2.20}$$

which gives:

$$||\mathbf{e}^k||_2 \leqslant \rho(\mathbf{H})^k \kappa(\mathbf{R}) ||\mathbf{e}^0||_2 \,. \tag{2.21}$$

For iteration matrices where the Eigenvectors are orthogonal, $\kappa(\mathbf{R}) = 1$ and the error bound reduces to:

$$||\mathbf{e}^k||_2 \leqslant \rho(\mathbf{H})^k ||\mathbf{e}^0||_2 \,. \tag{2.22}$$

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 (2.22).

2.3 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:

$$\tilde{\mathbf{x}} = \mathbf{x}_0 + \boldsymbol{\delta}, \;\; \boldsymbol{\delta} \in \mathcal{K} \,. \tag{2.23}$$

We bound this correction by asserting that the new residual, $\tilde{\mathbf{r}}$, be orthogonal to $\mathcal{L}$:

$$\langle \tilde{\mathbf{r}}, \mathbf{w} \rangle = 0,\ \forall \mathbf{w} \in \mathcal{L}\ . \tag{2.24}$$

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 2.1.

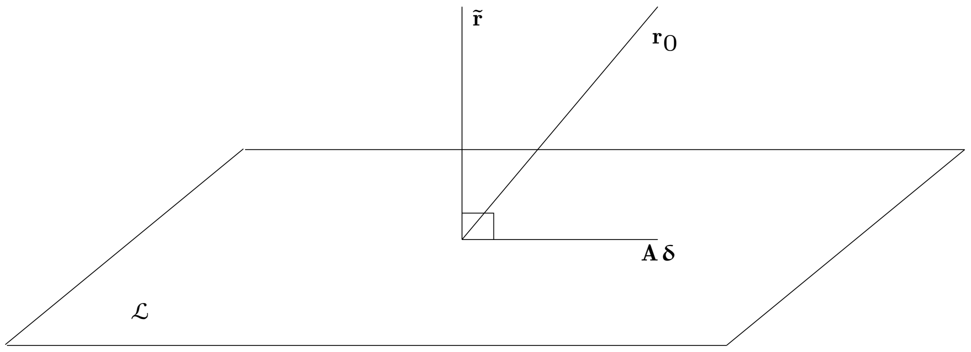

Figure 2.1: **Orthogonality constraint of the new residual with respect to $\mathcal{L}$.** *By projecting* $\mathbf{r}_0$ *onto the constraint subspace, we minimize the new residual by removing those components.*

From Figure 2.1 we then note that the following geometric condition must hold:

$$||\tilde{\mathbf{r}}||_2 \leqslant ||\mathbf{r}_0||_2,\ \forall \mathbf{r}_0 \in \mathbb{R}^{\mathrm{N}}\ , \tag{2.25}$$

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 (2.23), then $\boldsymbol{\delta}$ can instead be rewritten as:

$$\boldsymbol{\delta} = \mathbf{V}\mathbf{y},\ \forall \mathbf{y} \in \mathbb{R}^{\mathsf{N}} : \tag{2.26}$$

where $\mathbf{V}$ *projects* $\mathbf{y}$ onto $\mathcal{K}$. From the orthogonality constraint in Eq (2.24) it then follows that:

$$\mathbf{y} = (\mathbf{W}^{\mathsf{T}}\mathbf{A}\mathbf{V})^{-1}\mathbf{W}^{\mathsf{T}}\mathbf{r}_0 \,, \tag{2.27}$$

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:

$$\mathbf{r}^{\mathsf{k}} = \mathbf{b} - \mathbf{A}\mathbf{x}^{\mathsf{k}} \,, \tag{2.28a}$$

$$\mathbf{y}^{\mathsf{k}} = (\mathbf{W}^{\mathsf{T}}\mathbf{A}\mathbf{V})^{-1}\mathbf{W}^{\mathsf{T}}\mathbf{r}^{\mathsf{k}} \,, \tag{2.28b}$$

$$\mathbf{x}^{\mathsf{k}+1} = \mathbf{x}^{\mathsf{k}} + \mathbf{V}\mathbf{y}^{\mathsf{k}} \,, \tag{2.28c}$$

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 (2.25) 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.

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*:

$$\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0) = span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{m-1}\mathbf{r}_0\}\,, \tag{2.29}$$

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 (2.24) 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 (2.29). 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:

Algorithm 2.1 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
$\mathbf{y}_m \leftarrow argmin_y ||\beta \mathbf{e}_1 - \mathbf{H}_m\ \mathbf{y}||_2$
$\mathbf{x}_m \leftarrow \mathbf{x}_0 + \mathbf{V}_\mathbf{m} \mathbf{y}_m$
```

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 **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, 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.

2.4 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 2.1, 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 an implementations of such algorithms.

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:

$$\mathbf{y}[n] \leftarrow \mathbf{y}[n] + a * \mathbf{x}[n],\ \forall n \in [1, N_g]\ , \tag{2.30}$$

and locally of the form:

$$\mathbf{y}[n] \leftarrow \mathbf{y}[n] + a * \mathbf{x}[n],\ \forall n \in [1, N_l]\ , \tag{2.31}$$

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.

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 **y** and **x**, a single reduction is required such that:

$$d_l = \mathbf{y}_l \cdot \mathbf{x}_l, \ d_g = \sum_p d_l \, , \tag{2.32}$$

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:

$$||x||_{\infty,l} = \max_n \mathbf{y}[n], \ \forall n \in [1, N_l] \tag{2.33a}$$

$$||x||_{\infty,g} = \max_p ||x||_{\infty,l} \, . \tag{2.33b}$$

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.

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 2.2 that is partitioned on 3 processors. Each process owns a set of equations that correlate to the physical domain

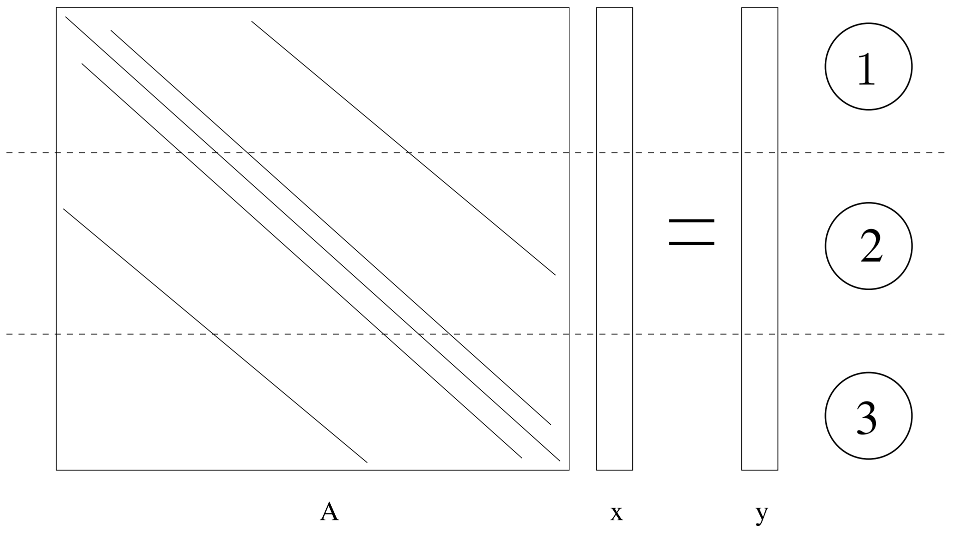

Figure 2.2: **Sparse matrix-vector multiply Ax = y operation partitioned on 3 processors.** *Each process owns a set of equations that correlates to its physical domain.*

of which it has ownership. 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 2.2. As shown in Figure 2.3, 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

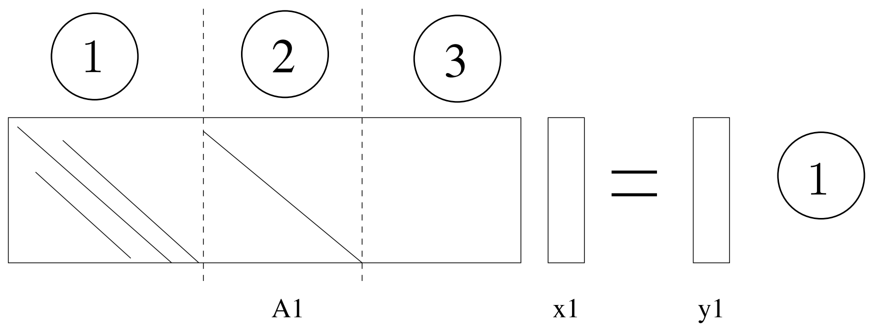

Figure 2.3: **Components of sparse matrix-vector multiply operation owned by process 1.** *The numbers above the matrix columns indicate the process that owns the piece of the global vector they are acting on. In order to compute its local components of the matrix-vector product, process 1 needs its 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,* $\mathbf{y}_1$*.*

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 2.3. Here, some of the data is intrinsically local, and some must be gathered from other processes using the partitioning data structures.

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 2.1. 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.

Chapter 3

Monte Carlo Solution Methods for Linear Systems

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 an enjoyable manuscript published in 1950 by Forsythe and Leibler (Forsythe and Leibler, 1950). In their outline, Forsythe and Liebler 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.

3.1 Preliminaries

We begin our discussion of Monte Carlo methods using these texts 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 (2.15) to define the following operator[1]:

$$\mathbf{H} = \mathbf{I} - \mathbf{A}\,, \tag{3.1}$$

[1]It should be noted that non-diagonal splittings have been recently explored in (Srinivasan, 2010) and have the potential to improve efficiency.

such that we are solving the system:

$$\mathbf{x} = \mathbf{H}\mathbf{x} + \mathbf{b} \,. \tag{3.2}$$

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

$$\mathbf{A}^{-1} = (\mathbf{I} - \mathbf{H})^{-1} = \sum_{k=0}^{\infty} \mathbf{H}^k \,, \tag{3.3}$$

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:

$$\mathbf{A}^{-1}\mathbf{b} = \sum_{k=0}^{\infty} \mathbf{H}^k\mathbf{b} = \mathbf{x} \,. \tag{3.4}$$

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 its i^{th} component:

$$x_i = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \dots \sum_{i_k}^{N} h_{i,i_1} h_{i_1,i_2} \dots h_{i_{k-1},i_k} b_{i_k} \,, \tag{3.5}$$

which can interpreted as a series of transitions between states,

$$\nu = i \rightarrow i_1 \rightarrow \cdots \rightarrow i_{k-1} \rightarrow i_k \,, \tag{3.6}$$

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

$$\mathbf{H} = \mathbf{P} \circ \mathbf{W}\,, \tag{3.7}$$

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 (3.5).

3.2 Direct Method

In the context of matrix inversion, a direct 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 direct method 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:

$$p_{ij} = \frac{|h_{ij}|}{\sum_j |h_{ij}|}\,. \tag{3.8}$$

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

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

the weight matrix are then defined by Eq (3.7) as:

$$w_{ij} = \frac{h_{ij}}{p_{ij}} . \tag{3.9}$$

It should be noted here that if **A** is sparse, then **H**, **P**, and **W** must be sparse as well by definition. Additionally, we only compute **P** and **W** from the non-zero elements of **H** as those components that are zero will not participate in the random walk. Doing so prevents an infinite weight from being generated in Eq (3.9) and eliminates unnecessary computations.

Using these matrices, we can then form the expectation value of the direct solution. For a given random walk permutation ν with k events, we define the weight of that permutation to be:

$$W_m = w_{i,i_1} w_{i_1,i_2} \cdots w_{i_{m-1},i_m} , \tag{3.10}$$

such that the weight of each transition event contributes to the total. The contribution to the solution from a particular random walk permutation is then:

$$X_\nu(i_0 = i) = \sum_{m=0}^{k} W_m b_{i_m} , \tag{3.11}$$

where $X_\nu(i_0 = i)$ signifies that the solution state in which we are tallying defines state i. We can interpret this precisely as before in that during the random walk we collect the source in the states that are visited and apply them to the solution tally. We then define the probability that a particular random walk permutation of k events will occur:

$$P_\nu = p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{k-1},i_k} . \tag{3.12}$$

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

$$E\{X(i_0 = i)\} = \sum_{\nu} P_{\nu} X_{\nu} \ , \tag{3.13}$$

which, if expanded, directly recovers the exact solution:

$$\begin{aligned} E\{X(i_0 = i)\} &= \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \dots \sum_{i_k}^{N} p_{i,i_1} p_{i_1,i_2} \dots p_{i_{k-1},i_k} w_{i,i_1} w_{i_1,i_2} \dots w_{i_{k-1},i_k} b_{i_k} \\ &= x_i \ , \end{aligned} \tag{3.14}$$

therefore providing an unbiased Monte Carlo estimator.

In cases where we seek only approximate solutions, we need only to perform a minimal number of random walks in order to generate an approximation for $\mathbf{x}$. If we are only to approximate the solution, we also need conditions by which we may terminate a random walk. We do this by noticing that the factors added to Eq (3.10) will become diminishingly small due to their definition in Eq (3.9) 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.

Estimator Variance

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

$$\sigma_i{}^2 = E\{X(i_0 = i) - (\mathbf{A}^{-1}\mathbf{b})_i\}^2 = E\{X(i_0 = i)^2\} - x_i^2 \ , \tag{3.15}$$

where the vector exponentials are computed element-wise. Inserting Eq (3.13) gives:

$$\sigma_i^2 = \Big(\sum_\nu P_\nu X_\nu^2 \Big)_i - x_i^2 \,, \tag{3.16}$$

and applying Eq (3.11):

$$\sigma_i^2 = \Big(\sum_\nu P_\nu \sum_{m=0}^{k} W_m^2 b_{i_m}^2 \Big)_i - x_i^2 \,. \tag{3.17}$$

Finally, expanding the transition probabilities yields the variance:

$$\sigma_i^2 = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \cdots \sum_{i_k}^{N} p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{k-1},i_k} w_{i,i_1}^2 w_{i_1,i_2}^2 \cdots w_{i_{k-1},i_k}^2 b_{i_m} - x_i^2 \,. \tag{3.18}$$

Using this definition for the variance, 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 (3.18):

$$(\mathbf{P} \circ \mathbf{W} \circ \mathbf{W})^k = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \cdots \sum_{i_k}^{N} p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{k-1},i_k} w_{i,i_1}^2 w_{i_1,i_2}^2 \cdots w_{i_{k-1},i_k}^2 . \tag{3.19}$$

If we assign $\mathbf{G} = \mathbf{P} \circ \mathbf{W} \circ \mathbf{W}$ as in Eq (3.7), we then have a new formulation for the variance:

$$\sigma_i^2 = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \cdots \sum_{i_k}^{N} g_{i,i_1} g_{i_1,i_2} \cdots g_{i_{k-1},i_k} b_{i_k}^2 - x_i^2 \,, \tag{3.20}$$

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

$$\mathbf{T} = \sum_{k=0}^{\infty} \mathbf{G}^k \,, \tag{3.21}$$

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

$$\sigma_i^2 = (\mathbf{Tb})_i - x_i^2 \,. \tag{3.22}$$

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

$$|\mathbf{H} \circ \mathbf{W}| \geqslant |\mathbf{H}|\ |\mathbf{W}| \,. \tag{3.23}$$

Using the norm property of the Hadamard product and Eq (3.7), we can define the norm of **W** as:

$$\frac{|\mathbf{H}|}{|\mathbf{P}|} \geqslant |\mathbf{W}| \,. \tag{3.24}$$

Choosing the infinity norm of the operator as defined in Eq (2.10), the row normalized probability matrix will yield a norm of 1 giving the following inequality for relating **G** and **H**:

$$|\mathbf{G}| \geqslant |\mathbf{H}|^2 \tag{3.25}$$

Using these relations to analyze Eq (3.22), we see that if $\rho(\mathbf{G}) > 1$, then the sum in Eq (3.21) will not converge and an infinite variance will arise as the elements of **T** become infinite in Eq (3.22). We must restrict **G** to alleviate this and therefore restrict **H** due to Eq (3.25) with $\rho(\mathbf{H}) < 1$ so that our expectation values for the solution may have a finite variance.

Reusing Random Walks

A downfall of the direct method is that a random walk must be initiated in each state of the solution vector in which the solution is desired due to the estimator definition in Eq (3.14). Because of this, a solution estimate in all states of a linear system may require a large number of random walks to be performed. Ji and Li recently proposed a variation of the direct method in which the random walk permutations in Eq (3.6) can be used to contribute to the solution tally in all visited states instead of the only starting state (Ji and Li, 2012). By doing this, they effectively reduce the number of random walks needed to sample all desired states in the system.

Their method is based on the observation that while a random walk transitions between states in the system, if every state it visited could be considered that starting state, then a series a contributions to the solution could be generated by a single sequence of random numbers while maintaining statistical independence between the samples. To demonstrate this, consider the following random walk sequence starting in state 2: $2 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5$. We can reuse these walks be recognizing that we also then have a random walk sequence starting in state 3 that can be applied to the state 3 estimator: $3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5$. We also have the same situation for state 1: $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$, and all future states. Using this sampling scheme , Ji and Li observed 1 to 2 orders of magnitude reduction over the standard direct method in the number histories required to achieve a specified variance in their test problems.

3.3 Adjoint Method

An alternative formulation for Monte Carlo matrix inversion is the adjoint method. We begin by defining the linear system adjoint to Eq (2.1):

$$\mathbf{A}^T \mathbf{y} = \mathbf{d}\,, \tag{3.26}$$

where $\mathbf{y}$ and $\mathbf{d}$ are the adjoint solution and source respectively and $\mathbf{A}^T$ is the adjoint operator. We can split this equation to mirror Eq (3.2) by defining the following inner product equivalence (Spanier and Gelbard, 1969):

$$\langle \mathbf{A}^T \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{A}\mathbf{y} \rangle\,. \tag{3.27}$$

With this statement we can then define the split equation:

$$\mathbf{y} = \mathbf{H}^{\mathsf{T}}\mathbf{y} + \mathbf{d}\,. \tag{3.28}$$

As was required for convergence with the direct method using Eq (3.2), the spectral radius of $\mathbf{H}$ must remain less than 1 as $\mathbf{H}^{\mathsf{T}}$ contains the same eigenvalues and therefore has the same spectral radius. From this definition it follows that:

$$\langle \mathbf{x}, \mathbf{d} \rangle = \langle \mathbf{y}, \mathbf{b} \rangle\,. \tag{3.29}$$

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 (3.28) as:

$$\mathbf{y} = (\mathbf{I} - \mathbf{H}^{\mathsf{T}})^{-1}\mathbf{d}\,, \tag{3.30}$$

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

$$\mathbf{y} = \sum_{k=0}^{\infty} (\mathbf{H}^{\mathsf{T}})^{k}\mathbf{d}\,. \tag{3.31}$$

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:

$$y_i = \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \dots \sum_{i_k}^{N} h_{i_k,i_{k-1}} \dots h_{i_2,i_1} h_{i_1,i} d_{i_k}\,. \tag{3.32}$$

We can readily build an estimator for the adjoint solution from this series expansion, but we instead desire the direct solution. We achieve this by using Eq (3.29) as a constraint. Here we have 2 unknowns, $\mathbf{y}$ and $\mathbf{d}$ and therefore we require a second constraint to close the system. As a second constraint we select:

$$\mathbf{d} = \boldsymbol{\delta}_i \,, \tag{3.33}$$

where the k^{th} component of the vector $\boldsymbol{\delta}_i$ is the Dirac delta function $\delta_{i_k,i}$. If we apply Eq (3.33) to our first constraint Eq (3.29), we get the following convenient outcome:

$$\langle \mathbf{y}, \mathbf{b} \rangle = \langle \mathbf{x}, \boldsymbol{\delta}_i \rangle = x_i \,, \tag{3.34}$$

meaning that if we compute the inner product of the direct source and the adjoint solution using a delta function source, we recover the direct solution we are looking for

In terms of particle transport, this adjoint method is equivalent to a traditional forward method. As a result of using the adjoint system, we modify our probabilities and weights using the *adjoint Neumann-Ulam decomposition* of **H**:

$$\mathbf{H}^T = \mathbf{P} \circ \mathbf{W} \,, \tag{3.35}$$

where now we are forming the decomposition with respect to the transpose of **H**[3]. 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:

$$p_{ij} = \frac{|h_{ji}|}{\sum_j |h_{ji}|} \,, \tag{3.36}$$

such that we expect to select a new state j from the current state in the random walk j by sampling column-wise. Per Eq (3.35), the transition weight is then defined as:

$$w_{ij} = \frac{h_{ji}}{p_{ij}} \,. \tag{3.37}$$

Using the decomposition we can then define an expectation value for the adjoint method. Given Eq (3.10) as the weight generated for a particular

[3]This is sometimes referred to as the adjoint form for real-valued matrices

random walk permutation as in Eq (3.6) and our result from Eq (3.34) generated by applying the adjoint constraints, the contribution to the solution in state i from a particular random walk permutation is then:

$$X_\nu = \sum_{m=0}^{k} W_m \delta_{i,i_m} , \tag{3.38}$$

where the Kronecker delta indicates that the tally contributes only in the current state and b_{i_0} will be the sampled source starting weight. Note here that the estimator in Eq (3.38) does not have a dependency on the source state as in Eq (3.38), providing a remedy for the situation in the direct method where we must start a random walk in each source state for every permutation such that we may compute a contribution 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:

$$E\{X\} = \sum_{\nu} P_\nu X_\nu \tag{3.39}$$

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

$$\begin{aligned} E\{X_j\} &= \sum_{k=0}^{\infty} \sum_{i_1}^{N} \sum_{i_2}^{N} \dots \sum_{i_k}^{N} b_{i_0} h_{i_0,i_1} h_{i_1,i_2} \dots h_{i_{k-1},i_k} \delta_{i_k,j} \\ &= x_j , \end{aligned} \tag{3.40}$$

therefore also providing an unbiased Monte Carlo estimate of the solution.

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*:

$$W_f = W_c b_{i_0} , \tag{3.41}$$

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.

3.4 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 $\exp(-N)$ (Evans et al., 2003). In much the same way as projection methods, Halton's method, sequential Monte Carlo, utilizes the adjoint Monte Carlo solver as a means of directly reducing the residual vector. He proposed the following iterative scheme as a solution to Eq (2.1)

$$\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k \,, \tag{3.42a}$$

$$\mathbf{A}\boldsymbol{\delta}^k = \mathbf{r}^k \,, \tag{3.42b}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \boldsymbol{\delta}^k \,, \tag{3.42c}$$

where the correction $\boldsymbol{\delta}$ is computed by the adjoint Monte Carlo method. 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. Therefore, if a good initial guess is not available, Halton's method is still characterized by poor performance.

3.5 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 and marginally better performance than conventional Krylov solvers. 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)

$$\mathbf{x} = (\mathbf{I} - \mathbf{A})\mathbf{x} + \mathbf{b} \,. \tag{3.43}$$

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

$$\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{A})\mathbf{x}^{k} + \mathbf{b} \,, \tag{3.44}$$

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:

$$\delta\mathbf{x}^{k} = \mathbf{x} - \mathbf{x}^{k} \,. \tag{3.45}$$

Subtracting Eq (3.44) from Eq (3.43) we get:

$$\delta\mathbf{x}^{k+1} = (\mathbf{I} - \mathbf{A})\delta\mathbf{x}^{k} \,. \tag{3.46}$$

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

$$\begin{aligned}\mathbf{A}\delta\mathbf{x}^{\mathrm{k+1}} &= (\mathbf{I}-\mathbf{A})(\mathbf{x}^{\mathrm{k+1}}-\mathbf{x}^{\mathrm{k}}) \\ &= \mathbf{r}^{\mathrm{k+1}}\,.\end{aligned} \tag{3.47}$$

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

$$\mathbf{x}^{\mathrm{k+1}} = (\mathbf{I}-\mathbf{A})\mathbf{x}^{\mathrm{k}} + \mathbf{b}\,, \tag{3.48a}$$

$$\mathbf{A}\delta\mathbf{x}^{\mathrm{k+1}} = \mathbf{r}^{\mathrm{k+1}}\,, \tag{3.48b}$$

$$\mathbf{x} = \mathbf{x}^{\mathrm{k+1}} + \delta\mathbf{x}^{\mathrm{k+1}}\,. \tag{3.48c}$$

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:

$$\mathbf{x}^{\mathrm{k+1/2}} = (\mathbf{I}-\mathbf{A})\mathbf{x}^{\mathrm{k}} + \mathbf{b}\,, \tag{3.49a}$$

$$\mathbf{r}^{\mathrm{k+1/2}} = \mathbf{b} - \mathbf{A}\mathbf{x}^{\mathrm{k+1/2}}\,, \tag{3.49b}$$

$$\mathbf{A}\delta\mathbf{x}^{\mathrm{k+1/2}} = \mathbf{r}^{\mathrm{k+1/2}}\,, \tag{3.49c}$$

$$\mathbf{x}^{\mathrm{k+1}} = \mathbf{x}^{\mathrm{k+1/2}} + \delta\mathbf{x}^{\mathrm{k+1/2}}\,, \tag{3.49d}$$

where the adjoint Monte Carlo method is used to generate the solution correction from the residual. 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 adjoint method to generate the correction.

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:

$$||\mathbf{r}||_\infty < \epsilon \; ||\mathbf{b}||_\infty \,, \tag{3.50}$$

where ϵ is a user-defined parameter. We therefore have 3 parameters to tune in an MCSA implementation: the number of Monte Carlo histories computed in the adjoint solve during each MCSA iteration, the weight cutoff for those histories, and the total MCSA convergence tolerance as specified by ϵ.

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. To achieve this, a Jacobi preconditioner, a form of left preconditioning, is used such that the preconditioning matrix $\mathbf{M}$ is:

$$\mathbf{M} = diag(\mathbf{A}) \,, \tag{3.51}$$

such that its application means we are instead solving the following linear system:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \,. \tag{3.52}$$

Next, we can apply MCSA to solve Eq (3.52):

$$\mathbf{x}^{k+1/2} = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{x}^{k} + \mathbf{M}^{-1}\mathbf{b}\,, \tag{3.53a}$$

$$\mathbf{r}^{k+1/2} = \mathbf{M}^{-1}\mathbf{b} - \mathbf{M}^{-1}\mathbf{A}\mathbf{x}^{k+1/2}\,, \tag{3.53b}$$

$$\mathbf{M}^{-1}\mathbf{A}\delta\mathbf{x}^{k+1/2} = \mathbf{r}^{k+1/2}\,, \tag{3.53c}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k+1/2} + \delta\mathbf{x}^{k+1/2}\,. \tag{3.53d}$$

Choosing Jacobi preconditioning with MCSA is advantageous for several reasons. First, $\rho(\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}) < 1$ is true for all $\mathbf{A}$ and is easy to formulate because the inversion of $\mathbf{M}$ is trivial. Second, because the adjoint Monte Carlo method used within MCSA to compute the correction operates on a linear problem with the preconditioned operator, then $\mathbf{H}$ in the adjoint 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, Jacobi preconditioning should always be performed, regardless of any other preconditioning that is applied to the system.

3.6 Parallelization of Stochastic Methods

For Monte Carlo methods, and in particular MCSA, to be viable at the production scale, scalable parallel implementations are required. In a general linear algebra context for discrete systems, this has not yet been achieved. Therefore, we will develop and implement parallel algorithms for these methods leveraging both the knowledge gained from the general parallel implementations of Krylov methods and modern parallel strategies for Monte Carlo as developed by the reactor physics community. In order to formulate a parallel MCSA algorithm, we recognize that the algorithm occurs in two stages, an outer iteration performing Richardson's iteration and applying the correction, and an inner Monte Carlo solver that is generating the correction via the adjoint method. The parallel aspects of both these components must be considered.

Domain Decomposition for Monte Carlo

As observed in the discussion on parallel Krylov methods, large-scale problems will surely have their data partitioned such that each parallel process owns a subset of the equations in the linear system. Given this convention, the adjoint Monte Carlo algorithm must perform random walks over a domain that is decomposed and must remain decomposed due to memory limitations. This naturally leads us to seek parallel algorithms that handle domain decomposition.

In the context of radiation transport, Brunner and colleagues provided a survey of algorithms for achieving this as implemented in production implicit Monte Carlo codes (Brunner et al., 2006). 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. The algorithms presented are a fully-locking synchronous scheme, an asynchronous-send/synchronous-receive pattern, a traditional master/slave scheme, and a modified master/slave scheme the implements a binary tree pattern for the global reduction type operations needed to communicate between the master and slave processes. They observed that the modified master/slave scheme performed best in that global communications were implemented more efficiently than those required by the asynchronous scheme. Furthermore, none of these schemes handled load-imbalanced cases efficiently. Such cases will be common if the source sampled in the Monte Carlo random walk is not isotropic and not evenly distributed throughout the global domain. It was noted that efficiencies were improved by increasing the frequency by which particle data was communicated between domain-adjacent processors. However, this ultimately increases communication costs. In 2009, Brunner extended his work by using a more load-balanced approach with a fully asynchronous communication pattern (Brunner and Brantley, 2009). Although the extended implementation was more robust and allowed for scaling to larger 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 Brunner to conclude that a combination of domain decomposition and domain replication could be used to solve some of these issues.

Multiple-Set Overlapping-Domain Decomposition

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 the 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 $\mathsf{O}(100)$ processors and if replicated $\mathsf{O}(1,000)$ times achieves simulation on $\mathsf{O}(100,000)$ processors, thus providing 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 $\mathsf{O}(100)$ 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 3.1. In this example, 3 particle histories are presented emanating

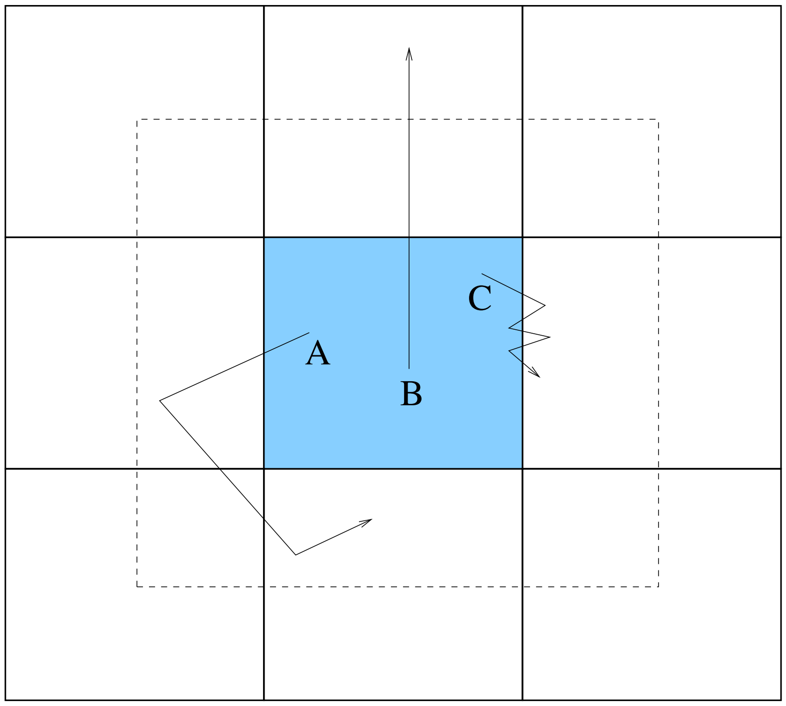

Figure 3.1: **Overlapping domain example illustrating how domain overlap can reduce communication costs.** *All particles start in the blue region of interest. The dashed line represents 0.5 domain overlap between domains.*

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 communications must occur in order to complete the particle A transport process. Using 0.5 domain overlap also easily eliminates cases such as the 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 3.1 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 constituted a 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 these results, this overlap approach, coupled with the multiple sets paradigm that will scale for existing parallel transport algorithms, provides a scalable Monte Carlo algorithm for today's modern machines.

Domain-to-Domain Communication

With Wagner's work, we observed that domain overlap implemented with a domain decomposed Monte Carlo setting reduces the amount of nearest neighbor communication that must occur during transport. This communication, expected to be expensive, was analyzed by Siegel and colleagues for idealized, load balanced situations without domain overlap (Siegel et al., 2012a). They arrive at a few empirical relationships and generate a numerical experiment that provides insight into the feasibility of large scale domain decomposed Monte Carlo as well as the relationship of communication to system parameters. In their work, they define a simple 3-dimensional system subdivded into Cartesian block domains with each processor containing a single domain. Given this geometry, a particle can either be absorbed in its current domain or be transported to one of the 6 adjacent domains owned by another process.

To quantify the number of particles that leak out of the local domain they define a leakage fraction, λ, as:

$$\lambda = \frac{\mathit{average}\ \#\ \mathit{of\ particles\ leaving\ local\ domain}}{\mathit{total\ of}\ \#\ \mathit{of\ particles\ starting\ in\ local\ domain}}\,. \tag{3.54}$$

For their studies, they assumed that the value of λ was dependent on the total cross section of the system and the same for all processes for simplicity and load balancing. Using this relationship and their simple geometry, they were able to relate the amount of compute time required to transport all source particles to the number of source particles, the leakage fraction, the latency and bandwidth of communication, the ratio of local to global geometry size, and the total cross section of the system.

What they found was that there was indeed an optimum ratio of local domain to global domain size that results in a minimum total compute time. They also found that this ratio was only dependent on the latency and bandwidth constraints of the parallel machine. We expect this because nearest neighbor communication patterns scale with the volume and surface sizes of the local domains. The larger the volume, the more compute time is spent within the local volume and therefore at some point you are limited by the latency of the system, or the time it takes to receive the data to operate on. However, at a certain point, when the surface of those volumes becomes large enough, the amount of data that must be communicated grows large enough that it may overcome the bandwidth limitations of the system, resulting in a decrease in performance. In their numerical experiments, they used the above geometry and leakage rates based on that computed for a reactor system for a single material problem with a total cross section equivalent to that of homogenized light water reactor. Given the optimal local domain size, they observed for load balanced situations that nearest neighbor communication costs during transport are a small fraction of total compute time, leading to the conclusion that domain decomposition is feasible for large-scale Monte Carlo calculations.

Load Balancing Concerns

Although domain decomposition was shown to be efficient in a perfectly load balanced situation in Siegel's work (Siegel et al., 2012a), careful consideration must be made for situations where this is not the case. Given the stochastic nature of the problem and lack of a globally homogeneous domain, parallel Monte Carlo simulations are inherently load imbalanced. Procassini and others worked to alleviate some of the load imbalances that are generated by both particle and spatial parallelism and are therefore applicable to the MSOD algorithm (Procassini et al., 2005). They chose a dynamic balancing scheme in which the number of times a particular domain was replicated was dependent on the amount of work in that domain (i.e. domains with a high particle flux and therefore more particle histories to compute require more work). In this variation, domains that require more work will be replicated more frequently at reduced particle counts in each replication. Furthermore, Procassini and colleagues noted that as the simulation progressed and particles were transported throughout the domain, the amount of replication for each domain would vary as particle histories began to diffuse, causing some regions to have higher work loads and some to have smaller work loads than the initial conditions.

Consider the example in Figure 3.2 adapted from Procassini's work. In this example, a geometry is decomposed into 4 domains on 8 processors with a point source in the bottom left domain. To begin, because the point

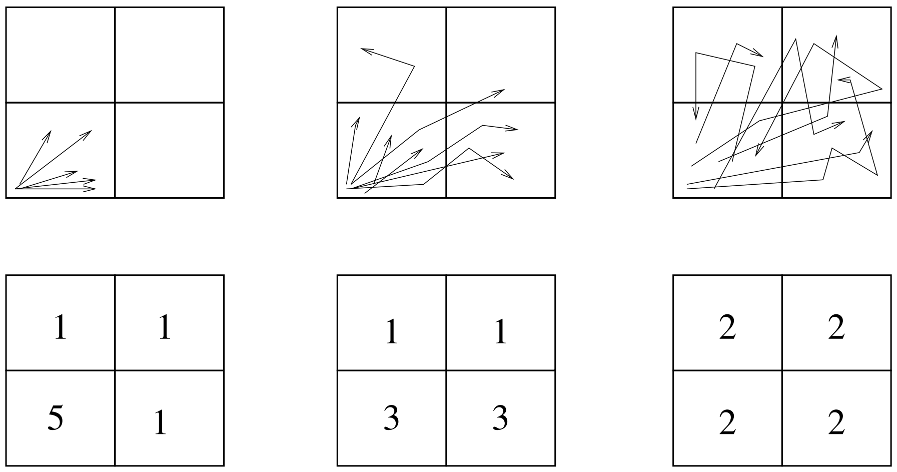

Figure 3.2: **Example illustrating how domain decomposition can create load balance issues in Monte Carlo.** *A domain is decomposed into 4 zones on 8 processors with a point source in the lower left zone. As the particles diffuse from the source in the random walk sequence as shown in the top row, their tracks populate the entire domain. As given in the bottom row, as the global percentage of particles increases in a zone, that zone's replication count is increased.*

source is concentrated in one domain, that domain is replicated 5 times such the amount of work it has to do per processor is roughly balanced with the others. As the particles begin to diffuse away from the point source, the amount of replication is adjusted to maintain load balance. Near the end of the simulation, the diffusion of particles is enough that all domains have equal replication. By doing this, load balance is improved as each domain has approximately equal work although each domain may represent a different spatial location and have a differing number of histories to compute. Compared to Wagner's work where the fission source was distributed relatively evenly throughout the domain, fixed source problems (and especially those that have a point-like source) like those presented in Procassini's work will be more prone to changing load balance requirements.

Reproducible Domain Decomposed Results

The 2006 work of Brunner is notable in that the Monte Carlo codes used to implement and test the algorithms adhered to a strict policy of generating identical results independent of domain decomposition or domain replication as derived from the work of Gentile and colleagues (Gentile et al., 2005). In Gentile's work, a procedure is given for obtaining results reproducible to machine precision for an arbitrary number of processors and domains. Differences can arise from using a different random number sequence in each domain and performing a sequence of floating point operations on identical data in a different order, leading to variations in round-off error and ultimately a non-identical answer. They use a simple example, recreated below in Figure 3.3, that illustrates these issues. In this example, the domain is decomposed on two processors with each processor owning one of the two zones. Starting with particle A, it is born in zone 1 and is transported to zone 2 where a scattering event occurs. Concerning the first reproducibility issue, if the same sequence of random numbers is not used to compute the trajectory from the new scattering event, we cannot expect to achieve the same result if the domain were not decomposed. If the numbers are different, the scattering event in zone 2 may keep the particle there or even eject it from the domain if the sequence were different. The second issue is demonstrated by adding another particle B that remains in the domain. In this case, an efficient algorithm will transport particle A on processor 1 until it leaves zone 1 and then transport particle B. Particle A will not renter the domain until it has been communicated to processor 2, processor 2 performs the transport, and it is communicated back to processor 1. If we are doing a track-length tally in zone 1, then we sum the tracks lengths observed in that zone. In the single processor, single zone case particle A would be transported in its entirety and then particle B transported. This would result in a tally sum with the following order of operations: $(((A1 + A2) + B1) + B2)$. If we were instead to use 2 processors, we would

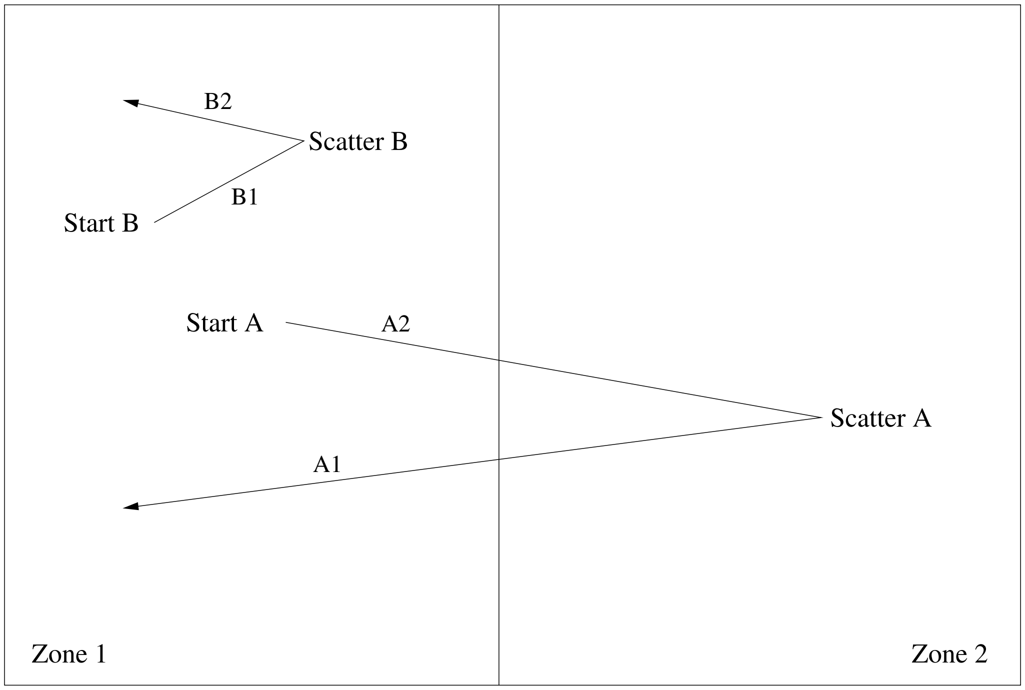

Figure 3.3: **Gentile's example illustrating how domain decomposition can create reproducibility issues in Monte Carlo.** *Both particles A and B start in zone 1 on processor 1. Particle A moves to zone 2 on processor 2 and scatters back to zone 1 while B scatters in zone 1 and remains there. A1 and A2 denote the track of particle A that is in zone 1 while B1 and B2 denote the track of particle B that is in zone 1.*

instead have the following order: $(((\mathsf{A}1 + \mathsf{B}1) + \mathsf{B}2) + \mathsf{A}2)$. In the context of floating point operations, we cannot expect these to have an identical result to machine precision as round-off approximations will differ resulting in non-commutative addition.

Procassini's solutions to these problems are elegant in that they require a minimal amount of modification to be applied to the Monte Carlo algorithm. To solve the first issue, in order to ensure each particle maintains an invariant random number sequence that determines its behavior regardless of domain decomposition, each particle is assigned a random number seed that describes its current state upon entering the domain of a new processor. These seeds are derived from the actual geometric location of the particle such that it is decomposition invariant. Non-commutative floating point operations are overcome by instead mapping floating point values to 64-bit integer values for which additions will always be commutative. Once the operations are complete, these integers are mapped back to floating point values.

Parallel Adjoint Method

We can take much of what was learned from the survey of parallel Monte Carlo methods for radiation transport and directly apply it to a parallel formulation of our stochastic linear solvers. Direct analogs can be derived from these works by noting that the primary difference between solving a linear system with Monte Carlo methods and 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 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 physical domain 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. Based on the results observed in the last decade of parallel Monte Carlo development, we can generate many practical questions that this type of work could answer.

Given a decomposed domain, per Wagner's work (Wagner et al., 2010) is clear that domain overlap significantly reduces the amount of communication required between adjacent domains as histories move to states that are not owned by the local processor. How much domain overlap is suitable for matrix inversion problems? Are we memory limited by large problems such that only so much overlap is feasible? Are there metrics related to the properties of the matrix including the eigenvalues and sparsity pattern such that we can provide guidelines for selecting the amount of overlap required? Furthermore, as Wagner and colleagues observed (Wagner et al., 2010), with marginal domain overlap for reactor problems, the percentage of histories leaving the local domain can easily be reduced to less than 1%. Evans and colleagues in their initial MCSA development typically used only 50 particle histories in order to compute a correction in an MCSA iteration (Evans et al., 2012). Per the Central Limit Theorem, that correlates to a statistical uncertainty of 14.1% in the correction, yet as compared to conventional Krylov solvers, MCSA achieved identical numerical results. If good convergence and numerically accurate solutions can be achieved with such a large uncertainty in the correction computation, then perhaps with enough domain overlap the minimal amount of histories that do transition to non-local states can be ignored and thus eliminate all communication in the parallel Monte Carlo adjoint solver transport sequence and create an embarrassingly parallel method. Given Gropp's work on parallel Krylov methods that we discussed in § 2.4 and the empirical results of Siegel (Siegel et al., 2012a), we know that these nearest neighbor computations between adjacent domains do not scale as well as global reduction operations and therefore we are improving scaling by eliminating them from the transport sequence. It will be important to determine if the expectation value bias in the MCSA correction generated by this approximation will remain within the bounds of the already high statistical uncertainty for unbiased estimates, thus providing numerically equivalent results.

From a domain replication perspective, this may be difficult to achieve with a production scale linear solver. Typically, memory is at a premium and therefore the more distinct domains available in the decomposition, the spatially finer and/or the numerically more accurate the discretization that can be implemented. How much replication is possible for large problems? How does replication facilitate parallel performance when coupled with domain overlap? How can we measure how much domain overlap is feasible? Replicating domains may therefore run into memory limitations for exceptionally large problems such that the operator, solution vector, and source vector must be copied in their entirety multiple times. From a resiliency standpoint, such an operation will be required, and therefore its performance and memory implications on conventional problems must be analyzed.

Parallel MCSA

With a parallel adjoint Neumann-Ulam solver implementation, the parallel implementation of the MCSA method will be trivial. Recall the MCSA iteration procedure outlined in Eq (3.49). In § 2.4 we discussed parallel matrix and vector operations as utilized in conventional Krylov methods. We utilize these here for the parallel MCSA implementation. 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 (3.50), 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. For this implementation, all of the issues that will be potentially generated by the parallel adjoint solver implementation will manifest themselves here as the quality of the correction will be of intense study.

In addition to parallel implementation and performance, MCSA's potential for aiding advancement in non-symmetric matrix solutions leads to a natural comparison with the GMRES algorithm. As both solvers are aimed at the same class of problem, we desire a set of metrics that will allow us to quantitatively compare the two. Given that the Krylov subspace maintained by GMRES can become large, do we benefit from a memory standpoint with an MCSA scheme in that no subspace is required? Does this benefit outweigh the fact that the linear operator must be explicitly formed in order to build the transition probabilities for the random walk sequence? For non-symmetric systems, does MCSA exhibit similar convergence properties to Krylov methods? If a Krylov methods build a subspace, can those memory savings in MCSA be used to implement domain replication in the adjoint solver? Such questions can be answered by a comparative study of the two solvers that controls the system size and the iterations required to converge.

Chapter 4

Monte Carlo Solution Methods for Nonlinear Systems

Nonlinear equation sets are a common occurrence in multiphysics 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, fully implicit 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 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:

$$||\mathbf{J}(\mathbf{u}^k)\delta\mathbf{u}^k + \mathbf{F}(\mathbf{u}^k)|| \leqslant \eta^k ||\mathbf{F}(\mathbf{u}^k)||\,, \tag{4.7}$$

where $\eta^k \in [0, 1)$ is defined as the *forcing term* at the k iterate. Eq (4.7) 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).

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 2, 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).

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:

$$\mathbf{J}(\mathbf{u})\mathbf{v} = \frac{\mathbf{F}(\mathbf{u}+\epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})}{\epsilon}\,, \tag{4.8}$$

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 (4.8) 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.

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:

$$\mathbf{J}(\mathbf{u}) = \sum_{i=1}^{N} \mathbf{Q}_i^T \mathbf{J}_k \mathbf{P}_i \,, \tag{4.9}$$

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 (4.9) 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.

4.3 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 3, 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.

Jacobian Storage vs. Subspace Storage and Restarts

To gauge the memory benefits of a FANM method over a Krylov-based scheme, we must look at the storage requirements of the Jacobian matrix as compared to the Krylov subspace vectors for sparse linear problems. Saad's text provides us relations for both of these cases (Saad, 2003). Beginning with the Jacobian storage, for sparse problems production linear algebra library implementations utilize *compressed row storage* to efficiently store the non-zero components of a sparse matrix while still allowing for all standard parallel matrix computations to be performed. Per § 2.4, recall that efficient parallel matrix-vector multiplications rely on processors knowing which other processors contain their neighboring matrix and vector elements. Typically this information is represented by a graph which must be stored along with the matrix elements themselves. To store the elements, consider the following sparse matrix:

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 8 & 0 & 0 & 0 \\ 4 & 5 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 3 & 7 & 0 & 2 \\ 0 & 0 & 0 & 4 & 9 & 0 \\ 0 & 0 & 0 & 0 & 9 & 1 \end{bmatrix} .$$

The compressed form of **A** would then be:

$$\begin{array}{llllllllllllllll} \mathbf{A}\ values: & 2 & 8 & 4 & 5 & 1 & 2 & 1 & 1 & 3 & 7 & 2 & 4 & 9 & 9 & 1 \\ column: & 1 & 3 & 1 & 2 & 4 & 2 & 3 & 5 & 3 & 4 & 6 & 4 & 5 & 5 & 6 \\ row\ start: & 1 & 3 & 6 & 9 & 12 & 14 & 16 \end{array}$$

where the first row contains the non-zero values of the matrix **A** starts at, the second row contains the column index of those values, and the third row contains the index of the A value and column index rows that each matrix row starts at ending with the start of the next row. For this simple example, we actually break even by storing 36 elements in the full matrix and 36 pieces of data in the compressed format. However, for large, sparse matrices there is a significant storage savings using the compressed format.

In his discussion on orthogonalization schemes for Krylov subspaces, Saad provides us with space and time complexities for these operations. To find a solution vector $\mathbf{x} \in \mathbb{R}^{N \times N}$ with m Krylov iterations (and therefore $m+1$ subspace vectors) the storage requirement is then $(m+1)N$ for most common orthogonalization techniques. Using our simple example above (even though we did not see any gains in storage over the full matrix representation), if it takes 10 GMRES iterations using Arnoldi orthogonalization to converge to a solution, then a total of 66 pieces of data are required to be stored instead of 36 for the explicitly formed matrix case (we do not consider graph storage here for comparison as both methods will require the graph for parallel operations). For larger sparse matrices, this disparity in memory requirements will be even greater, especially for ill-conditioned systems that require many GMRES iterations to converge.

Parallel FANM Implementation

Like the parallelization of the MCSA method described in § 3.6, 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 § 4.2. 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 § 4.3, 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.

Chapter 5

Research Proposal

To develop the parallel MCSA and FANM methods, a research plan is proposed to drive their development and quantify their merits and feasibility as compared to more conventional methods for linear and nonlinear problems. We seek to rigorously verify both of these methods and their implementation and will describe the means by which that will be accomplished. Furthermore, the numerical experiments performed will be constructed within a software framework that will permit an agile development strategy. In this chapter we describe that experimental framework and the progress achieved to date by its use. In addition, we will a formulate a plan for verifying the new Monte Carlo methods and prepare a set of numerical experiments that will determine some of their properties. Finally, we select a large-scale challenge problem designed to potentially demonstrate the effectiveness of our new methods in the context of real-world, production level applications for nuclear reactor analysis.

5.1 Experimental Framework

For all of our numerical experiments, we need to readily be able to do two things: generate linear and nonlinear systems in parallel. To do this effectively requires many software components to be implemented including linear algebra data structures, a parallel communication framework, mesh generation and partitioning, application of boundary and initial conditions, automatic differentiation technology, and other high-level framework technologies. These development of these components, however, are not the focus of this work but rather the necessary machinery needed for the experimental apparatus. We therefore choose an environment in which all of these tools are readily available in the form of production software implementations and that will demonstrate scalability in high performance computing environments. This environment will be based on the mathematical framework of the finite element method and will permit an algorithm-oriented design for the parallel stochastic solvers we will develop.

Finite Element Formulation

Choosing a finite element formulation for our physics problems permits us flexibility from both a mathematical and software development standpoint. We briefly review the *finite element method* as presented in Zienkiewicz's text (Zienkiewicz et al., 2005). In general, we are concerned with solving a set of differential equations:

$$\mathbf{A}(\mathbf{u}) = \mathbf{0},\ \forall \mathbf{u} \in \Omega \tag{5.1a}$$

$$\mathbf{B}(\mathbf{u}) = \mathbf{0},\ \forall \mathbf{u} \in \Gamma\,, \tag{5.1b}$$

where Ω is the domain over which the set of differential equations is defined and Γ is the boundary of that domain. We then have a set of differential equations, $\mathbf{A}(\mathbf{u})$, in residual form that represent the physics on the domain and the set $\mathbf{B}(\mathbf{u})$ that is representative of those physics on the boundary. As an alternative, we can instead state these differential equations in an equivalent integral form via a conservation statement:

$$\int_\Omega \mathbf{v}^T \mathbf{A}(\mathbf{u}) d\Omega + \int_\Gamma \bar{\mathbf{v}}^T \mathbf{B}(\mathbf{u}) d\Gamma = \mathbf{0}\,, \tag{5.2}$$

where $\mathbf{v}$ and $\bar{\mathbf{v}}$ are arbitrary, finite-valued vectors that satisfy the integral relation and the constraints in Eq (5.1).

If the n^{th}-order derivative of $\mathbf{u}$ contains discontinuities (i.e. is not smooth), then the integral form presented in Eq (5.2) is restricted in that only derivatives up to order $n+1$ may appear in the functions $\mathbf{A}(\mathbf{u})$ and $\mathbf{B}(\mathbf{u})$. We can relax this restriction by integrating Eq (5.2) by parts to arrive at the *weak form* of the differential equations:

$$\int_{\Omega} \mathbf{C}(\mathbf{v}^{\mathsf{T}})\mathbf{D}(\mathbf{u})d\Omega + \int_{\Gamma} \mathbf{E}(\bar{\mathbf{v}}^{\mathsf{T}})\mathbf{F}(\mathbf{u})d\Gamma = \mathbf{0}\,, \tag{5.3}$$

where the new functional form now has lower-order derivatives in $\mathbf{D}(\mathbf{u})$ and $\mathbf{F}(\mathbf{u})$, thus relaxing the smoothness requirement.

In our experimental framework, we will formulate all of our problems using the weak form of the differential equations as in Eq (5.3). Doing so allows us to leverage modern finite element assembly engine software with embedded automatic differentiation technology. Using the Panzer library in the Trilinos scientific computing framework (Notz and Pawlowski, 2010; Heroux et al., 2005), function evaluation routines in the form of Eq (5.3) are generated to represent the finite element problem and boundary conditions. Per Bartlett's automatic differentiation work and an associated parallel linear algebra framework, the equation sets for all physics implemented are assembled into the nonlinear system. By leveraging the general linear algebra framework, we are able to abstract away those concepts in the Monte Carlo solvers implementation and instead focus on an algorithm-oriented approach for the solver implementation (Musser and Stepanov, 1994).

5.2 Progress to Date

To date, progress has been made on development of the experimental framework based on the Panzer library. Using this framework, several preliminary objectives have been achieved including an implementation of the MCSA algorithm using a general linear algebra framework and initial analysis of its performance using both the direct and adjoint Neumann-Ulam methods. Here we report the results of this work and its relationship with future work.

Generalization of MCSA for Linear Problems

Previous implementations of MCSA had an entirely physics-based approach (Evans and Mosher, 2009; Evans et al., 2012). In these works, the transition probabilities and weights were generated directly from the discretized differential equations instead of a general linear operator. With this type of approach, transition probabilities and weight formulations must be rederived for each type of physics implemented. To reach the broader physics community and facilitate deployment for more complicated physical systems and multiple physics problems, MCSA would benefit from an implementation that utilizes the general operator form of a linear system as in Eq (2.1). Operating on the general form of the linear system will also greatly simplify the implementation of our numerical experiments as different equations sets and boundary conditions can be rapidly implemented within the Panzer library to generate a complex array of linear problems.

Generalizing MCSA for all linear problems has been achieved with both the direct and adjoint Neumann-Ulam methods as the Monte Carlo solver by leveraging the Epetra library within Trilinos (Heroux et al., 2005). Using Epetra (and its variants) we have access to several features that will enhance future development and experimentation. First, Epetra employs a compressed storage format for sparse matrices as outlined in § 4.3. This will permit efficient memory use and allow for the savings we expect by implementing the FANM method. Second, Epetra is fully parallelized in that the parallel graph structures and decomposed storage formats are utilized to domain decompose both the operator and vectors in the linear system. These data structures provide access to the required parallel matrix-vector operations outlined in § 2.4. In addition, the parallel Monte Carlo methods we will implement for the Neumann-Ulam methods as outlined in § 3.6 will be greatly facilitated and perhaps even enabled by these data structures.

Finally, the Epetra framework and its variants are commonplace in many production level physics codes and therefore such an implementation will provide an easy path to incorporating an MCSA or FANM solver into those codes for further experimentation and analysis.

Direct vs. Adjoint Analysis

Preliminary work has been completed to characterize the behavior of the direct and adjoint Monte Carlo solvers with the MCSA method.[1] The aim of this initial work was to determine which of the two Monte Carlo solvers provided the best MCSA performance. The MCSA method defined in Eq. (3.49) uses the adjoint method to estimate the error instead of the direct method outlined in § 3.2. To demonstrate the effectiveness of the adjoint method over the direct method within the context of MCSA, we choose the 2-dimensional time-dependent Poisson equation as a simple model problem:

$$\frac{\partial \mathbf{u}}{\partial t} = \nabla^2 \mathbf{u}\,. \tag{5.4}$$

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-order five-point stencil,

$$\nabla_5^2 = \frac{1}{\Delta^2}[u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}]\,, \tag{5.5}$$

and a fourth-order nine-point stencil,

$$\begin{aligned}\nabla_9^2 = \frac{1}{6\Delta^2}[&4u_{i-1,j} + 4u_{i+1,j} + 4u_{i,j-1} + 4u_{i,j+1} + u_{i-1,j-1} \\ &+ u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} - 20u_{i,j}]\,,\end{aligned} \tag{5.6}$$

[1]The contents of this section have been taken directly from my contributions to (Evans et al., 2012)

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:

$$\mathbf{A}\mathbf{u}^{n+1} = \mathbf{u}^{n} \,. \tag{5.7}$$

Both the stencils will be used to vary the size and density of the sparse linear system in Eq. (5.7).

A timing and convergence study is used to demonstrate the effectiveness of the adjoint method 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 grid size, 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, 50 random walk permutations were computed per MCSA iteration with 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. All computations presented in this section were completed on a 2.66 GHz Intel Core i7 machine with 8 GB 1067 MHz DDR3 memory. Figure 5.1 gives the CPU time needed for each case to converge in seconds, and Fig. 5.2 gives the number of iterations needed for each case to converge relative to the problem size.

We see clearly in Fig. 5.1 that the using the adjoint solver with MCSA results in several orders of magnitude speedup over the direct solver while the number of iterations required to converge is of a similar scale. We expect this for several reasons. First, with an equivalent number of histories specified for both solvers and a system of size $N \times N$, the direct solver will compute $N \times N$ random walks per permutation while the adjoint solver will only compute one. 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, because the random walk sequence

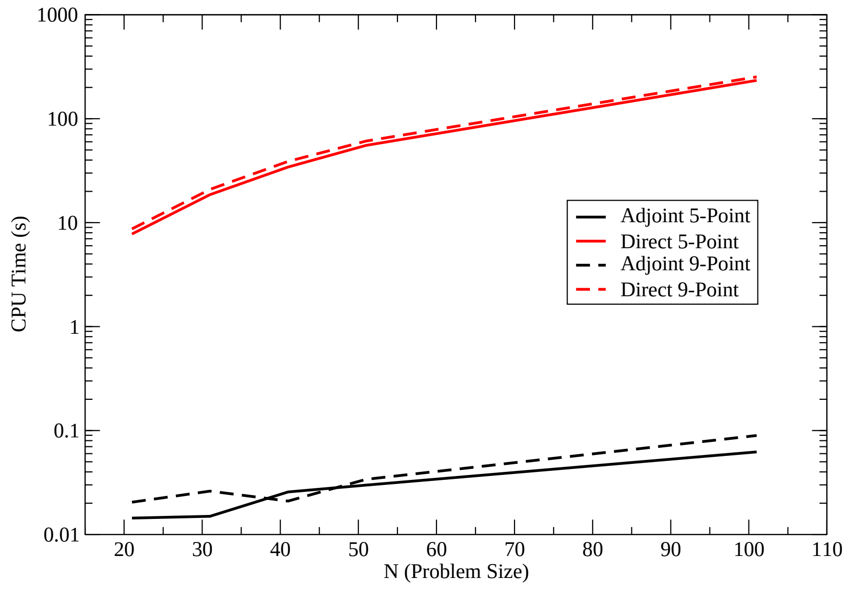

Figure 5.1: **CPU Time (s) to converge vs. Problem Size (N for an N $\times$ N square mesh).** *Both the adjoint and direct solvers are used with the five point and nine point stencils. A significant CPU time speedup is noted with the adjoint method due to the reduced number of random walk events.*

contributes to the state in which it currently resides, fewer histories are necessary to compute contributions from all the nonzero parts of the source. Initially, this may not seem like a fair comparison; however, we see from Fig. 5.2 that the number of iterations required to converge is approximately the same and therefore the extra computations performed by the direct method do not improve the error estimation. Therefore, although the total number of random walks per iteration is in fact larger for the direct method, the solution computed at each iteration is no better than that computed by the adjoint method.

As an additional comparison, the convergence behavior of MCSA can

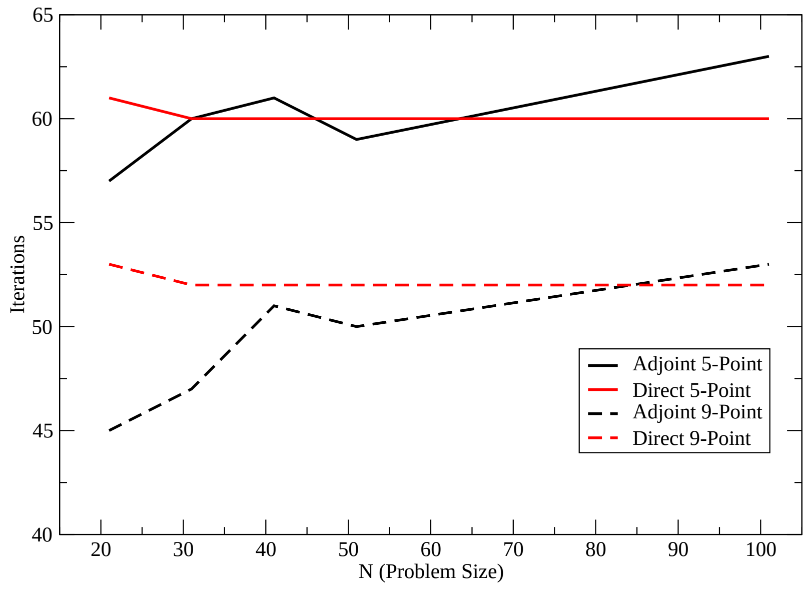

Figure 5.2: **Iterations to converge vs. Problem Size (N for an $\mathsf{N} \times \mathsf{N}$ square mesh).** *Both the adjoint and direct solvers are used with the five-point and nine-point stencils. Both methods give similar iteration behavior.*

be analyzed using both 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. (3.49) was collected at each iteration for a fixed problem size. Figure 5.3 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. Second, we note that both the direct and adjoint methods have approximately equivalent convergence behavior, leading to CPU time being the dominating factor in method selection.

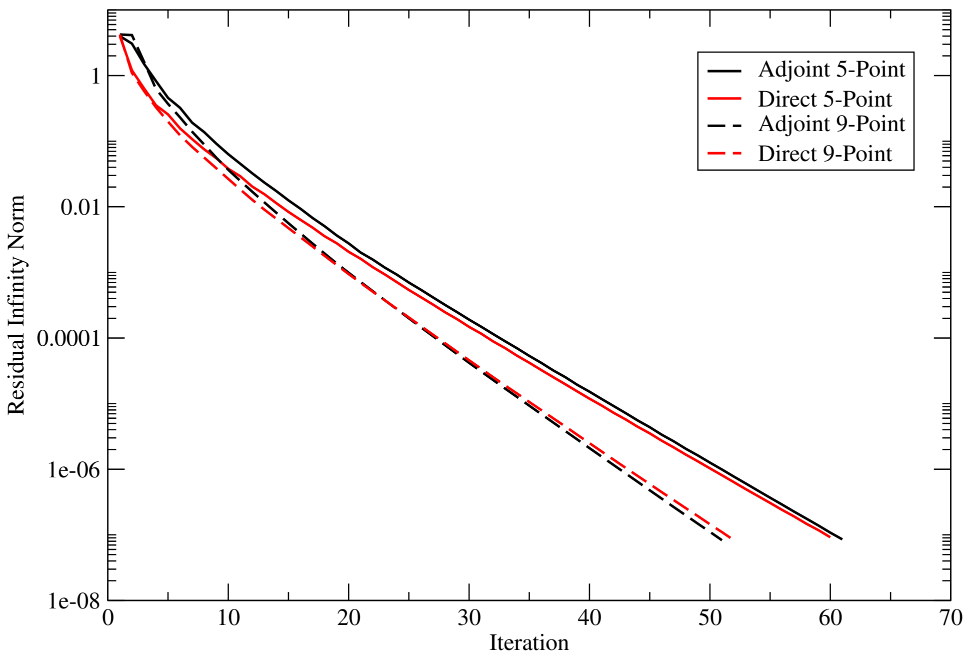

Figure 5.3: **Infinity norm of the solution residual vs. iteration number for a problem of fixed size.** *Both the adjoint and direct solvers are used with the five point and nine point stencils.*

MCSA Application to Finite Element Problems

The Epetra-based MCSA implementation has enabled the solution of finite element problems leveraging the Panzer finite element assembly engine, providing an initial basis for our experimental framework. To demonstrate this, consider the 2-dimensional steady-state heat equation:

$$\boldsymbol{\nabla} \cdot k \boldsymbol{\nabla} T = Q \,, \tag{5.8}$$

where k is the thermal conductivity, T is the temperature, and Q is the thermal source. We can write the heat equation in weak form using Eq (5.3):

$$\int_{\Omega} \boldsymbol{\nabla}^T v k \boldsymbol{\nabla} T d\Omega - \int_{\Omega} v Q d\Omega - \int_{\Gamma} v \bar{q} d\Gamma - \int_{\Gamma} v k \frac{\partial T}{\partial n} d\Gamma = 0 \,, \tag{5.9}$$

where the boundary conditions have now been incorporated. We can interpret the last two integral terms as the contributions from a fixed boundary flux source and the flux through the domain boundary. This weak formulation can then be directly implemented within the finite element assembly engine along with the appropriate boundary conditions and source terms. For this example, we choose a square Cartesian grid with boundary conditions as shown in Figure 5.4. A value of 2.0 is used for k with no thermal

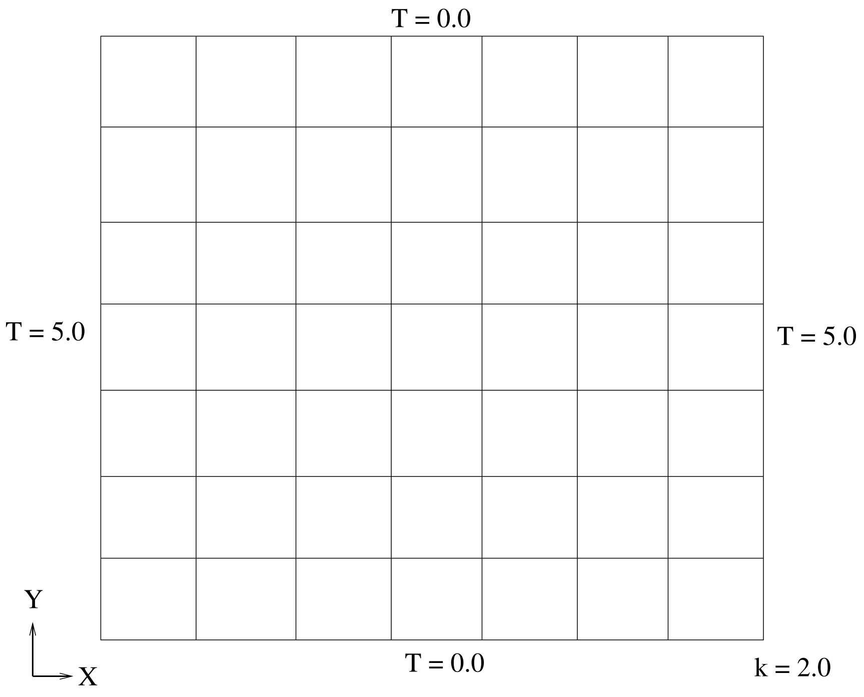

Figure 5.4: **Problem setup for finite element solution to 2D heat equation.** *Dirichlet conditions are set for the temperature on all 4 boundaries of the Cartesian grid with thermal conductivity set to 2.0. No thermal source was present.*

source and second-order basis functions are chosen for the finite element form. This problem was solved using Panzer to assemble the finite element system and Jacobi preconditioned MCSA as the linear solver with the adjoint Neumann-Ulam method as the Monte Carlo solver. The results of this calculation are presented in Figure 5.5. In the results, we see the

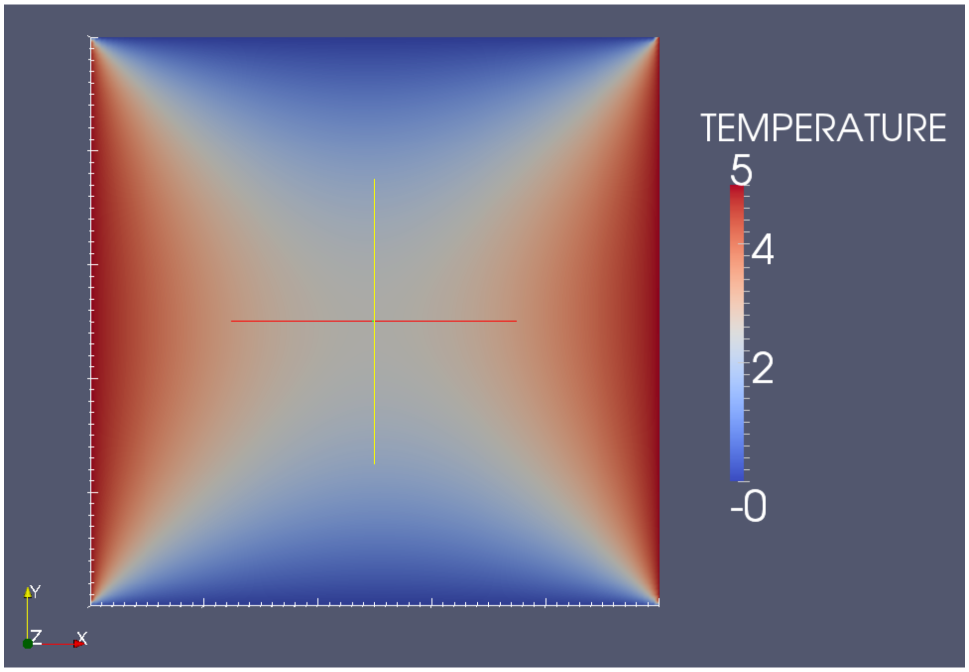

Figure 5.5: **Finite element solution to the 2D steady-state heat equation using MCSA.** *A* 100×100 *Cartesian grid was used with Jacobi preconditioning applied to the linear system.*

characteristic parabolic solution profiles of the heat equation corresponding to the correct Dirichlet condition. Based on this work, a finite element formulation generated by the Panzer framework can be readily solved with a general MCSA implementation. In addition, the above problem can already be solved in parallel with conventional Krylov methods. Therefore, once a parallel MCSA implementation is complete, this framework can be used to generate those experiments and immediately compare to Krylov methods. In addition, extension to nonlinear model problems requires only a few small additions to the framework, providing the remainder of the tool set we need for our analysis.

5.3 Monte Carlo Methods Verification

As we perform our numerical experiments, the results from the new tools that we are using must be verified against benchmark results. These benchmark tests should allow use to verify that the new stochastic solvers that we develop generate the expected numerical results in parallel for linear and nonlinear problems that have a known, well-documented solution. In addition, as we explore parallel strategies for the Monte Carlo methods and introduce bias or non-reproducible results into our solution estimators, these verification problems will allow us to determine if that cost is acceptable. To verify the linear solvers, we will choose the above 2-dimensional heat equation problem in Eq (5.8) on a rectilinear grid for which analytical solutions are available. As a complement to comparison with the analytical solutions, we may also check those results against production implementations of Krylov solvers. For the nonlinear solvers, we employ a new set of equations with readily available benchmark solutions. In both cases, we verification will be determined by comparing the locations of global minima and maxima in the solution as well as comparison of solution vector norms with respect to a specified tolerance.

FANM Verification

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:

$$\rho \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \cdot \mathbf{T} - \rho \mathbf{g} = \mathbf{0} \tag{5.10a}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{5.10b}$$

$$\rho C_p \mathbf{u} \cdot \nabla T + \nabla \cdot \mathbf{q} = 0\,, \tag{5.10c}$$

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 (5.10a) provides momentum transport, Eq (5.10b) provides the mass balance, and Eq (5.10c) provides energy transport with viscous dissipation effects neglected. In addition, we close the system with the following equations:

$$\mathbf{T} = -P\mathbf{I} + \mu[\nabla \mathbf{u} + \nabla \mathbf{u}^T] \tag{5.11a}$$

$$\mathbf{q} = -k\nabla T\,, \tag{5.11b}$$

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.

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 5.6 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.

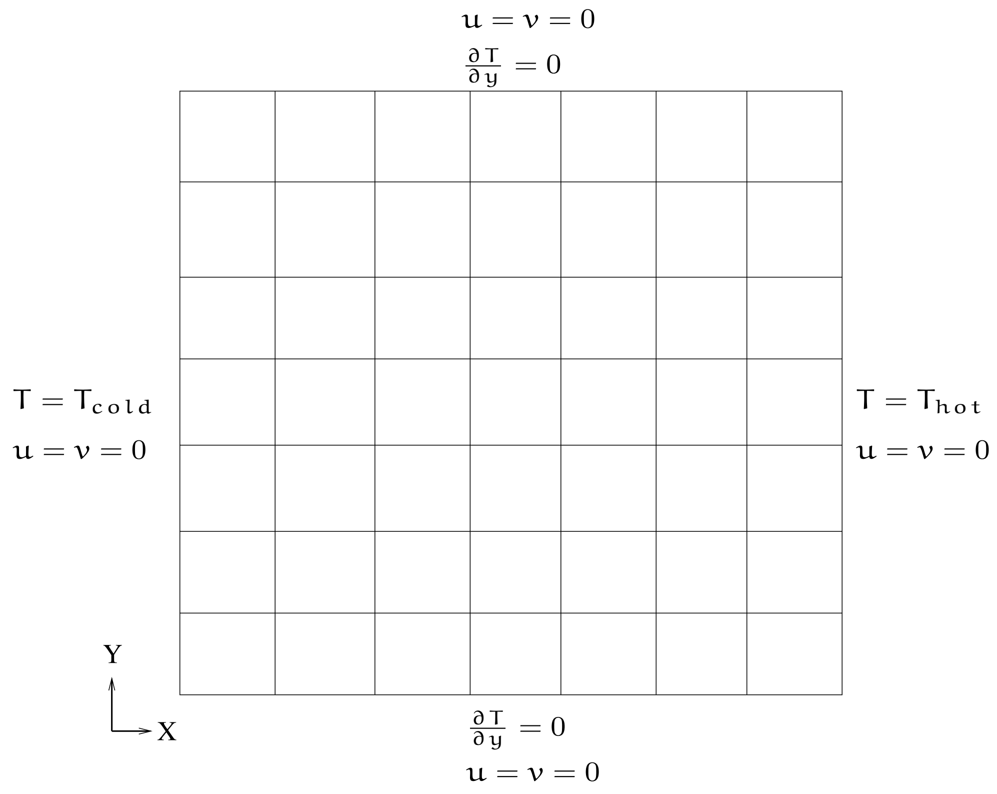

Figure 5.6: **Problem setup for the natural convection cavity benchmark.** *Dirichlet conditions are set for the temperature on the left and right while Neumann conditions are set on the top and bottom of the Cartesian grid. The temperature gradients will cause buoyancy-driven flow. Zero velocity Dirichlet conditions are set on each boundary. No thermal source was present.*

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 (5.10a). In Shadid's work, Rayleigh numbers of up to 1×10^6 were used for this benchmark on a 100×100 square mesh grid.

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 5.7 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 (5.10a). Shadid also used a 100×100 grid for this benchmark problem with Reynolds numbers up to 1×10^4.

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 5.8. 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

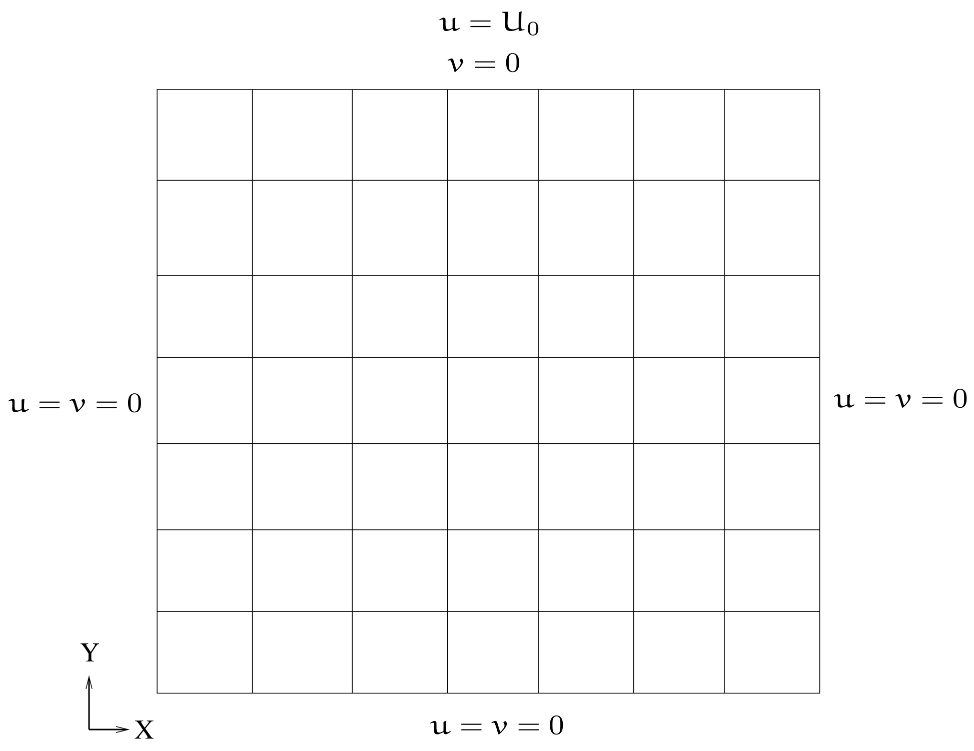

Figure 5.7: **Problem setup for the lid driven cavity benchmark.** *Dirichlet conditions of zero are set for the velocity on the left and right and bottom while the Dirichlet condition set on the top provides a driving force on the fluid.*

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.

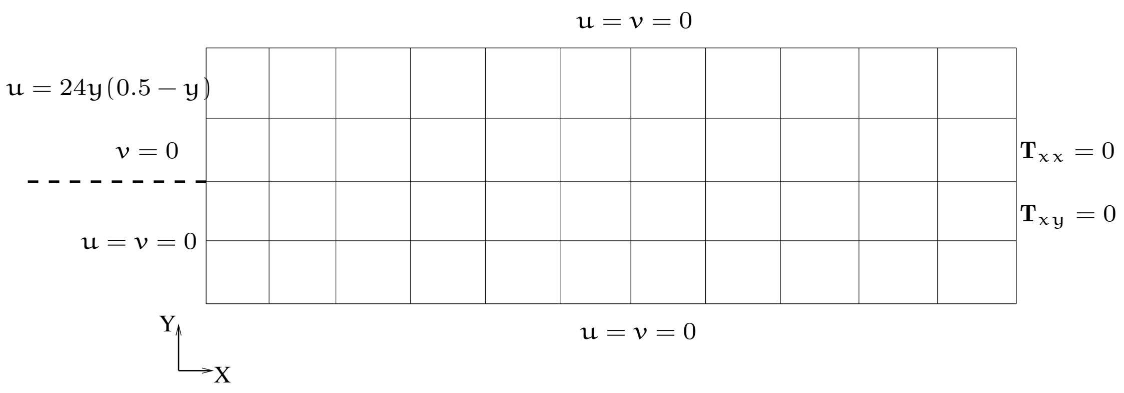

Figure 5.8: **Problem setup for the backward facing step benchmark.** *Zero velocity boundary conditions are applied at the top and bottom of the domain while the outflow boundary condition on the right boundary is represented by zero stress tensor components in the direction of the flow. For the inlet 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.*

5.4 Proposed Numerical Experiments

Using the verification problems for the linear and nonlinear solvers as a testing environment, we propose a set of experiments that will be used to characterize several qualities of parallel Monte Carlo solvers. At the end of Chapter 3, we posed several research questions related to the parallelization of the adjoint Neumann-Ulam method and the MCSA method while in Chapter 4 we posed questions related to the FANM method. We will generate numerical experiments from these questions by dividing them into 4 categories: domain overlap studies for the parallel Neumann-Ulam method, domain replication studies for the parallel Neumann-Ulam method, parallel performance and numerical accuracy studies for the MCSA method, and parallel performance and numerical accuracy studies for the FANM method.

Domain Overlap Studies for the Parallel Neumann-Ulam Method

Characterizing and understanding the domain overlap behavior of a parallel Neumann-Ulam method will be critical to efficient implementations of other solvers that will be built on top of these methods. In the reactor physics work that we reviewed, the amount domain overlap used could be directly correlated to physical parameters of the system, namely the dimension of repeated geometric structures and the mean free path of neutrons in the system. We therefore seek analogous measurements for the general linear system. These measurements will characterize how far from its starting state a random walk sequence moves as a function of properties of the linear system. Our experimental framework will allow us to fix various properties of the system while varying others including the size of the linear system, its spectral radius, its sparsity, its asymmetry, and other parameters related to the Eigenvalues, conditioning, and structure of the linear operator. By varying the domain overlap and measuring parallel performance with respect to these properties, we can develop guidelines for setting the overlap with respect to measurable properties of the linear operator. In addition, varying the amount of overlap will allow us to measure the additional memory costs incurred.

Domain Replication Studies for the Parallel Neumann-Ulam Method

Employing domain replication with the Neumann-Ulam method will be an exercise in batch statistics. By using replication, we also expect to incur an additional memory overhead. As noted in previous work, domain replication was coupled with domain overlap in order to improve load balancing and scalability in the implementations. As replication will likely be required to solve future resiliency issues and may even by required by this work in order to achieve scalability, its parallel performance must also be measured. We will therefore measure the memory and performance overhead incurred by varying the amount of domain replication leveraged by the Neumann-Ulam solver with respect to a fixed linear system.

Parallel Performance and Numerical Accuracy Studies for the MCSA Method

A parallel MCSA implementation and our numerical test framework will allow us to compare its performance to Krylov methods and allow us to experiment with the way we combine it with the Neumann-Ulam solver. As discussed in § 3.6, previous work with MCSA used a correction generated by the Neumann-Ulam method with a fairly large uncertainty near 14%. We hypothesized that if we can identify the amount of overlap in our Neumann-Ulam experiments required to eliminate most transitions between adjacent domains owned by different processors, then perhaps transport to adjacent domains can be ignored. This will result in a biased estimator for the MCSA correction that cannot be reproduced, however, these studies will aim to determine if that bias and lack of reproducibility affects the numerical results and reproducibility of the MCSA method, yielding a critical outcome for this work. Furthermore, we will use the experimental framework to measure the convergence rates for non-symmetric systems and memory usage of the MCSA algorithm and compare them to conventional Krylov methods using the same framework.

Parallel Performance and Numerical Accuracy Studies for the FANM Method

Aside from resiliency motivations, the primary driver behind the FANM method is reduced memory pressure due to the lack of additional storage beyond that of the linear system in the MCSA method. The nonlinear benchmark problems we have chosen will let us measure this memory usage and compare it to conventional Newton-Krylov methods. Furthermore, by varying the strength of the nonlinearities in those problems (effectively by increasing the amount of convective transport in the system), the benchmarks will become more difficult to solve and require even more iterations to converge. This will give us a means by which to vary the complexity of the system which we can then compare to memory usage. Additionally, with the finite element assembly framework we can choose functional discretizations of varying accuracy, allowing us to vary the degrees of freedom in the system and therefore also vary the memory required for each problem.

5.5 Proposed Challenge Problem

Our numerical experiments will provide us with a set of metrics giving best-performance Monte Carlo solver parameters for problems of different types. As an additional bonus, these experiments will let us further develop new parallel implementations of the Monte Carlo methods in order to improve their performance and scalability. With these advancements in hand, we look forward towards applying them to large-scale problems in reactor physics. The Consortium for Advanced Simulation of Light Water Reactors (CASL) is a Department of Energy Modeling and Simulation Hub focused on predictive multiphysics modeling of nuclear reactors (U.S. Department of Energy, 2011). Their efforts are focused on enabling nuclear reactor analysis and development using today's supercomputing technology to achieve higher power uprates, higher fuel burn-up, and predictive accident scenario analysis. As a means of driving their research and development, CASL has worked with the nuclear industry to identify challenge problems of critical importance to achieving these goals.

To meet some of the high fidelity modeling requirements of these challenge problems, the Drekar multiphysics code is being developed at Sandia National

Laboratories for fluid flow and conjugate heat transfer problems (Pawlowski et al., 2012). Drekar is being used to analyze complex problems including the grid-to-rod-fretting (GTRF) phenomena observed in light water reactor fuel rods. GTRF occurs due to the turbulent flow of water through the subchannels of the reactor core. This turbulence causes the nuclear fuel pins to vibrate and the fuel cladding to wear against structures in the reactor, causing localized hot spots and ultimately cladding failure. Reducing these instances of failure and understanding how they occur through predictive multiple physics simulation of fluid flow and heat transfer will permit more robust fuel assembly designs. In addition, the turbulence studies that Drekar performs will allow for improved flow patterns to be generated in the assembly subchannels, allowing for better heat transfer between the nuclear fuel and the water, ultimately allowing for a more efficient reactor.

Drekar is a massively parallel physics framework built upon the same Panzer library as well as other open-source tools that we are leveraging in our own experimental framework. Furthermore, its multiphysics formulation utilizes Newton-Krylov methods to generate a fully consistent solution scheme for the Navier-Stokes equations with various turbulence models for fluid simulation as well as the heat conduction problem for simulation in solid materials. Performance metrics show scaling to $\mathsf{O}(100,000)$ cores on leadership class machines with $\mathsf{O}(1 \times 10^9)$ degrees of freedom in these simulations. Given these features, Drekar will be an excellent framework in which we can pose our own challenge problem for the FANM and newly parallelized MCSA methods proposed by this research. At the time our preliminary numerical experiments are completed for characterizing the parallel FANM and MCSA implementations, we propose implementing them within the Drekar framework and performing the equivalent of its largest GTRF multiphysics problem to date, most likely to be a fuel-assembly sized problem. Doing so will allow us to not only implement our new methods in a production physics code and apply it to a problem of great interest to the nuclear community, but it will also allow us to compare its performance and scaling to the Newton-Krylov methods currently leveraged within Drekar. If successful, we expect to demonstrate improved computational performance from a memory usage and/or an algorithmic scaling perspective.

5.6 Conclusion

We have proposed the research and development of novel, massively parallel Monte Carlo methods for the solution of linear and nonlinear problems. These new methods aim to enhance the solution of multiple physics problems of great national interest. Current progress has shown that these methods can indeed be expanded into a general linear algebra framework that permits their application to a broad range of problems. Additional analysis has also been completed that begins to characterize their performance in different configurations. Leveraging this work, we have defined a multiphysics framework based on the finite element method and have demonstrated that these Monte Carlo methods can be used effectively to solve the linear systems generated by these discretizations.

To guide our development of parallel Monte Carlo methods, we have devised several classes of numerical experiments using our multiphysics framework that will provide insight into their properties and performance for linear and nonlinear problems. Furthermore, these experiments aim to provide metrics that will give an indication of both solver performance for various problem types and the type of parallel decomposition required to effectively solve these problems. Domain decomposition, domain replication, and their combination will be of great interest for the analysis of parallel Monte Carlo methods while algorithm scalability and memory usage will be of interest for the MCSA and FANM methods that leverage them.

We culminate our research and development with a challenge problem by applying these new Monte Carlo methods in the Drekar multiphysics code. With Drekar, we look forward to providing a new and robust solution method at the production scale. Using these tools and a leadership class computing platform, we will provide performance measurements for the Monte Carlo solvers and compare them to conventional methods currently leveraged within Drekar. Given the potential to improve computational performance in production environments such as Drekar, these new methods have the potential to help meet the high fidelity modeling and simulation needs of the nuclear community.

references

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.

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.

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.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.

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, 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.

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).

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.

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*.

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.