DEVELOPMENT OF METHODS FOR HIGH PERFORMANCE COMPUTING APPLICATIONS OF THE DETERMINISTIC STAGE OF COMET CALCULATIONS

A Dissertation
Presented to
The Academic Faculty

by

Kyle Eugene Remley

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Nuclear and Radiological Engineering

Georgia Institute of Technology

August 2016

DEVELOPMENT OF METHODS FOR HIGH PERFORMANCE COMPUTING APPLICATIONS OF THE DETERMINISTIC STAGE OF COMET CALCULATIONS

Approved by:

Dr. Farzad Rahnema, Advisor
George W. Woodruff School
Georgia Institute of Technology

Dr. Tom Morley
School of Mathematics
Georgia Institute of Technology

Dr. Bojan Petrovic
George W. Woodruff School
Georgia Institute of Technology

Dr. Alireza Haghighat
Virginia Polytechnic and State University

Dr. Dingkang Zhang
George W. Woodruff School
Georgia Institute of Technology

Date Approved: July 13, 2016

To my lovely life.

ACKNOWLEDGEMENTS

Major milestones in one's life are rarely achieved without the support and guidance of multiple people, and this thesis is no exception. I would like to thank my friends and family who have supported and inspired me during my tenure at Georgia Tech, as well as the people who have helped me become who I am today.

First, I'd like to thank my advisor, Dr. Farzad Rahnema for his support, guidance, and encouragement during my tenure as a graduate student. Further, I would like to thank him for pushing me to get involved with research at CRMPL at Georgia Tech and to swiftly complete my graduate studies. I would also like to thank the members of my committee – Dr. Bojan Petrovic, Dr. Dingkang Zhang, Dr. Tom Morley, and Dr. Alireza Haghighat – for participating in the critique of my work.

In addition, I'd like to thank my wonderful colleagues in the CRMPL Lab at Georgia Tech. The relationships that I have formed during our long hours in the lab are among my most treasured memories. My fellow CRMPLers Daniel Lago, Gabriel Anthony Kooreman, Ryan Hon, Alex Huning, Chris "Chaps" Chapman, Stefano Terlizzi, and Drew Johnson certainly helped me through the many rough times in research as a much needed morale boost, and I can only hope that I returned the favor. I'd like to apologize for all the times I had "plans" on days when there were lab outings. The ones I actually managed to make it to were some of my favorite times in graduate school.

I must also mention my family, specifically my parents Keith and Belinda Remley, for bribing me with Pokemon cards for good grades when I was young. Had they not instilled in me a persistent motivation to achieve, I would not know where I am today.

Finally, I'd like to thank my friends Nathan Le and Nick Breen. Without their support and companionship through my many years in undergraduate and graduate school, I would be a much sadder man than I am today. I'm happy to be great friends with the both of you, and I hope our connections continue on for many years in the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Coarse Mesh Radiation Transport (COMET) method is a reactor physics method and code that has been used to solve whole core reactor eigenvalue and flux distribution problems. A strength of the method is its formidable accuracy and computational efficiency. COMET solutions are computed to Monte Carlo accuracy on a single processor in a runtime that is several orders of magnitude faster than stochastic calculations. However, with the growing ubiquity of both shared and distributed memory parallel machines and the desire to extend the method to allow for coupling to multiphysics and on-the-fly response generation, serial implementations of COMET calculations will become less desirable. It is under this motivation that an implementation for a parallel execution of deterministic COMET calculations has been developed. COMET involves inner and outer iterations; inner iterations involve local calculations that can be carried out independently, making the algorithm amenable to parallelization. However, considerations for decomposing a problem and the distribution of data must be made. To allow for efficient parallel implementation of a distributed algorithm, changes to response data access and sweep order are made, along with considerations for communications between processors. The parallel code is implemented on several variants of the C5G7 benchmark problem to assess the scalability of the algorithm, and it is found that problems with larger numbers of coarse meshes increase the scalability of the code, which is an encouraging result. The code is further tested for full core reactor problems, where extremely efficient

wall clock times (on the order of minutes) for solutions are achieved. Finally, application of the parallel code to novel implementations of COMET (e.g., problems with high flux expansions) is discussed.

CHAPTER 1

INTRODUCTION

Since its inception, the reactor physics community has considered efficient 3-D full core neutronics calculations as an important goal. This desire looms even larger in today's environment of heavy regulation of nuclear power, as fewer reactors are built for research purposes; instead, greater emphasis is placed on modeling and simulation to accomplish design goals. However, the sheer number of unknowns involved in classical (e.g., fine mesh) methods have limited the applicability of deterministic solution methods in accomplishing this goal. Stochastic methods have been shown to provide full-core eigenvalue solutions with the benefit of exact phase space treatment of neutron transport, but these methods are often computationally prohibitive, as reducing uncertainties in calculations and source convergence leads to long runtimes.

However, as computers have improved, so have these solution methods' efficacies in solving reactor physics problems. With growing computational capability through increasing power and memory, both deterministic and stochastic solution methodologies have increased their capabilities in tackling full core neutronics calculations. A particularly important advancement in computing applications has been the implementation of parallel algorithms. Both stochastic and deterministic codes have implemented solution methods that involve many processors working on partitions of a computational task rather than a single

processor carrying out calculations. This advancement is viewed as the prevailing strategy for scientific computing as a whole, as large computing systems with many processors become more commonplace, and the reactor physics community is no exception in using this strategy.

1.1 The COMET Method as a Way to Compute Whole-core Solutions

In addition to improving reactor physics calculations through increasing computing power, the community has also sought advanced methods for reactor physics calculations. For instance, hybrid methods are considered an attractive option as an alternative to deterministic and stochastic solution methods to facilitate full core neutronics calculations. The Coarse Mesh radiation Transport, or COMET, method developed at the Georgia Institute of Technology is a stochastic-deterministic method that has been shown to provide eigenvalue and flux distribution solutions to 3-D reactor core benchmark problems in full geometric heterogeneity [1-2]. The method has been benchmarked against several different reactor types, including light and heavy water reactor types such as PWR, BWR, and CANDU [3], as well as prismatic configurations, including a sodium cooled fast reactor [4] and a gas cooled reactor [5].

The COMET method is response-based. By decomposing a problem into a number of non-overlapping subvolumes, called coarse meshes, an eigenvalue problem for the full reactor core can be transformed into a system of local fixed-source problems coupled about the interface angular fluxes. If the interface angular fluxes and the global eigenvalue are known, this decomposition solves the

problem exactly. However, interface fluxes and eigenvalue are not known, so a way to proceed is with an angular flux expansion. Responses (e.g., resulting outgoing angular fluxes and pin fission density) to incoming boundary angular fluxes with phase space determined by an assumed expansion function are calculated for every unique coarse mesh in a precalculation. In order to preserve the heterogeneity in the problem, this precalculation is done stochastically. Once computed, these responses are stored in a library and are used to solve the decomposed system of fixed source problems via a deterministic iterative algorithm, allowing for a solution to the global problem to be found.

1.2 Motivation and Goal

An attractive feature of the COMET method is its formidable computational efficiency. Like Monte Carlo methods, COMET calculations are able to solve whole-core neutronics problems; however, COMET runtimes are several orders of magnitude smaller than that of stochastic calculations. Maintaining and increasing efficiency of the method has been an active area of research. To this end, various acceleration methods have been implemented in the deterministic algorithm, which are seen to significantly improve the computational efficiency [2]. In addition, techniques that adaptively select expansion order have been explored to improve efficiency of the method [6-9].

The goal of this study was to develop and asses the gains in computational efficiency of a parallel implementation of the deterministic iterative algorithm for COMET calculations. Particularly, efficient whole-core computations are desired

with the use of parallel computing. The work of this thesis builds on the research of increasing computational efficiency of the COMET method to applications that involve parallel computing. A parallel implementation of the deterministic algorithm is necessary to adapt the COMET method to the ubiquity of parallel machines to maintain and increase its computational efficiency in future implementations; as deterministic and stochastic methods have benefitted from parallel computing, so should the COMET method.

By extending deterministic COMET calculations to many processors, COMET will be better suited for taking advantage of novel (e.g., on-the-fly) response generation techniques as well as coupling to multiphysics, such as lattice depletion, both of which are current areas of research [10-11]. Both response generation and lattice depletion methods take place in parallel environments, so coupling the deterministic COMET solution to these calculations in a parallel environment increases efficiency in ways such as data sharing.

CHAPTER 2

BACKGROUND AND THEORY

In this section, the fundamental theory of neutron transport – the neutron transport equation – as well as various methods for analysis of phenomena will be discussed. While classical deterministic and stochastic methods will be mentioned, the primary focus of this section will be on the COMET method. For a more thorough review of classical deterministic and stochastic methods, the reader is encouraged to refer to Lewis and Miller [12]. Emphasis is placed on the theory and derivation of the global problem solution since response generation and deterministic solution procedure are implementation dependent.

2.1 The Neutron Transport Equation

The equation describing the steady-state phase space distribution for neutrons in a large heterogeneous volume V is given by the neutron transport equation:

$$\hat{\Omega}\cdot\nabla\psi(\vec{r},\hat{\Omega},E)+\sigma(\vec{r},E)\psi(\vec{r},\hat{\Omega},E)=\int_0^\infty dE'\int_{4\pi}d\hat{\Omega}'\sigma_s(\hat{\Omega}',E'\rightarrow\hat{\Omega},E)\,\psi(\vec{r},\hat{\Omega}',E')$$

$$+\frac{\chi(\vec{r},E)}{4\pi k}\int_0^\infty dE'\,\nu\sigma_f(\vec{r},E')\int_{4\pi}d\hat{\Omega}'\psi(\vec{r},\hat{\Omega}',E'), \qquad (1)$$

where ψ is the angular flux, $(\vec{r},\hat{\Omega},E)$ is the space-angle-energy phase space, $\chi(\vec{r},E)$ is the spectrum of fission neutron emission, σ, σ_s, and σ_f are total, scatter, and fission cross sections, ν is number of neutrons emitted per fission, and k is the

global eigenvalue. To determine the distribution of neutrons, equation (1) is paired with an appropriate boundary condition, given by

$$\psi(\vec{r}_b, \hat{\Omega}, E) = B\psi(\vec{r}_b, \hat{\Omega}', E'), \hat{n} \cdot \hat{\Omega} < 0, \hat{n} \cdot \hat{\Omega}' > 0, \ \vec{r}_b \in \partial V. \tag{2}$$

In equation (2), ∂V denotes the system boundary, $\vec{r}_b$ indicates a position along the boundary, $\hat{n}$ is the unit outward normal, and B is a boundary condition operator. The neutron transport equation (1) is long-winded with many terms. As such, it is often useful to write equation (1) in terms of operator notation:

$$H\psi(\vec{r}, \hat{\Omega}, E) = \frac{1}{k} F\psi(\vec{r}, \hat{\Omega}, E). \tag{3}$$

The transport H and fission F operators are defined as

$$\begin{cases} H = \hat{\Omega} \cdot \nabla + \sigma(\vec{r}, E) - \int_0^\infty dE' \int_{4\pi} d\hat{\Omega}' \sigma_s(\hat{\Omega}', E' \to \hat{\Omega}, E) \\ F = \frac{\chi(\vec{r}, E)}{4\pi} \int_0^\infty dE' \, \nu\sigma_f(\vec{r}, E') \int_{4\pi} d\hat{\Omega}' \end{cases}. \tag{4}$$

The boundary value problem (1-2) is a very difficult problem solve. Except in the most simplified of cases, solutions cannot be found analytically. We seek to find useful solutions for realistic reactor problems, so these simplified case solutions are often inadequate. As a result, we must approximate the neutron transport equation and seek numerical solutions.

2.2 Classical Methods

There are two main families of methods employed in solving the neutron transport equation: deterministic and stochastic solution methods. Both have seen much development and use in transport calculations; however, the methods also encounter considerable difficulties. This subsection provides a brief review of both types of methods.

Deterministic Solution Methods

Deterministic methods solve the boundary value problem (1-2) via discretization of phase space, defining a system of linear equations from this discretization, and then solving this system simultaneously. In the eigenvalue case, solving a system (inverting a matrix) is coupled with another solution method, typically a power iteration. However, sometimes, more sophisticated methods are utilized [13].

Arguably the most important discretization employed in deterministic methods is the multigroup approximation. In this approximation, solutions are found for a discrete number of group fluxes ψ_g rather than a continuous energy spectrum for the true angular flux ψ. This approximation leads to a system of integro-differential equations of the form

$$H\psi_g(\vec{r},\hat{\Omega}) = \frac{1}{k}F\psi_g(\vec{r},\hat{\Omega}) \tag{5}$$

which are coupled via the scattering and fission terms of each of the equations. This discretization has significant effect on accuracy, as it requires precomputation of group constants (e.g., cross sections) that are not known in their exact form before a true solution is found. How well these group constants relate to the problem being solved as well as number of energy groups used in a calculation greatly affects fidelity of the calculation. However, significant issues are memory and computation time – a more accurate calculation employing a large number of energy groups will lead to a large utilization of these resources, which are often limited.

Other discretizations to the transport equation depend upon method and implementation. A common angular discretization scheme is the discrete ordinates approximation [12], where the transport equation is solved for only discrete directions, and the integrals in the fission and scattering terms of the transport equations are approximated via quadrature. Spatial discretizations include finite difference, finite element, and characteristic methods [12,14-15]. As with the multigroup approximation, finer discretizations lead to more accurate answers. Again, however, this puts significant strain on available computing power and memory. The sheer number of unknowns generated in deterministic neutron transport calculations (as many as billions of unknowns) present a significant challenge to these methods.

Stochastic Solution Methods

In addition to deterministic methods, stochastic (e.g., Monte Carlo) methods have seen significant development. In contrast to deterministic tools, phase space is treated exactly in a Monte Carlo transport calculation. Instead of discretizing the transport equation, a stochastic method uses random numbers to simulate the transport of individual particles in a reactor. After a sufficiently large number of particles have been simulated, the mean behavior of neutrons in a reactor can be determined.

An advantage of these methods is that they can be used to model problem geometry explicitly, and compute global eigenvalue quickly. For Monte Carlo calculations, global eigenvalue is simply given by the ratio of fission sources between cycles:

$$k^i = \frac{F^i}{F^{i-1}}. \quad (6)$$

Typically, this is a fast calculation once the fission source is converged because computation is a single tally that occurs over a whole problem. However, the fission source must be converged first since it is not known before a calculation, which can sometimes be a significant computational hurdle.

Stochstic methods do suffer from some drawbacks. Pointwise angular fluxes cannot be computed with these methods; instead, quantities (e.g., angular flux) averaged over discretized phase space (e.g., direction, energy, or space) or part of the phase space (e.g., scalar flux) are most often calculated. This formal limitation does not hurt the method, as a quantity such as scalar flux is typically a more useful quantity in reactor calculations than angular flux. However, desire for detailed knowledge of the spatial variation of scalar flux in a reactor core requires relatively fine meshing. This can lead to large variances in tallies and statistical noise, as simulated particles may not travel through tally zones in a sufficiently large number. The main remedy to this issue is running more particles in a calculation; however, this, coupled with large numbers of tallies, can lead to infeasibly long computational runtimes.

2.3 The COMET Method

To remedy the issues surrounding classical deterministic and stochastic methods for solving the neutron transport equation, advanced methods are constantly being investigated. The focus of this study is on the COMET method developed at Georgia Institute of Technology [1-2]. COMET is a response matrix

method, which is a very powerful tool that can circumvent much of the computational expense of classical solution methods by shifting cost to a precalculation. In addition, source convergence issues which can plague Monte Carlo calculations, are circumvented by limiting the stochastic calculation to small (coarse mesh) fixed source problems. Given a set of precomputed data, solutions to even whole-core problems can be found with excellent computational efficiency.

Domain Decomposition

Suppose that we wish to compute the flux distribution and global eigenvalue of a reactor core given by the boundary value problem (1-2). The COMET method will decompose the problem volume V into a system of non-overlapping subvolumes V_i, henceforth called coarse meshes. For each coarse mesh, the following transport equation and boundary condition are assumed to hold:

$$\begin{cases} H\varphi_i(\vec{r},\hat{\Omega},E) = \frac{1}{k}F\varphi_i(\vec{r},\hat{\Omega},E) \\ \varphi_i^-(\vec{r}_{is},\hat{\Omega},E) = \psi(\vec{r}_{is},\hat{\Omega},E),\ \vec{r}_{is} \in V_{is} \end{cases}. \tag{7}$$

In the boundary value problem (7), φ_i is the angular flux within the i^{th} coarse mesh, and the incoming angular flux into surface s, indicated by the "-" superscript, is equivalent to the global angular flux at that surface for that mesh. In addition, it is assumed that the eigenvalue in the boundary value problem (7) is fixed at the global value. Given these assumption, the system (7) becomes a fixed-source problem. Further, if these assumptions are true, then the decomposition introduces no approximation to the calculation.

Introduction of a Flux Expansion

However, the global flux on coarse mesh surfaces and global eigenvalue are not known *a priori*. Therefore, a method to proceed is via a flux expansion [1-2]. Incoming and outgoing angular fluxes are assumed to be a superposition of a known complete set of orthogonal functions:

$$\varphi_i^{+/-}(\vec{r}_{is}, \widehat{\Omega}, E) = \sum_m J_{s,m}^{i,+/-} \Gamma_m \ (\vec{r}, \widehat{\Omega}, E). \tag{8}$$

In equation (8), the "+/-" superscript indicates outgoing or incoming flux, respectively. Γ_m are the expansion functions in moment m, and $J_{s,m}^{i,+/-}$ are the outgoing and incoming expansion coefficients for surface s in moment m for the i^{th} coarse mesh.

Due to the linearity of the transport equation, the i^{th} coarse mesh angular flux can be constructed as

$$\varphi_i = \sum_s \sum_m J_{s,m}^{i,-} R_{s,m}^i, \tag{9}$$

where $R_{s,m}^i$ is known as a surface-to-volume response function. This response function is defined as the solution to the equation

$$HR_{s,m}^i(\vec{r}, \widehat{\Omega}, E) = \frac{1}{k} F R_{s,m}^i(\vec{r}, \widehat{\Omega}, E) \tag{10}$$

with k fixed at the global value and coupled to the boundary condition

$$R_{s,m}^i(\vec{r}_{is}, \widehat{\Omega}^-, E) = \begin{cases} \Gamma^m(\vec{r}_{is}, \widehat{\Omega}^-, E) \ for \ \vec{r} \in \partial V_{is} \\ 0, \ otherwise \end{cases}. \tag{11}$$

It seen that $R_{s,m}^i$ is the angular flux within the volume V_i responding to the incident flux Γ^m, hence the term surface-to-volume response function.

In theory, any expansion functions can be used in this formalism. However, some requirements on the expansion functions are sensible if reasonable accuracy

with a relatively low expansion order is desired. Two such requirements are that the 0th order expansion function represents the isotropic flux and that the partial currents are conserved regardless of use of expansion order. Let the following inner product be defined:

$$(u, v) = \int dE \int_{\partial V_{is}} d\vec{r} \int_{\hat{n}_{is}^{+/-} \cdot \hat{\Omega} > 0} d\hat{\Omega} (\hat{n}_{is}^{+/-} \cdot \hat{\Omega}) uv. \tag{12}$$

It is then required that the expansion coefficients satisfy the following orthogonality condition:

$$\left(\Gamma_m , \Gamma_{m'}\right) = A_m \delta_{mm'} \tag{13}$$

where A_m is a constant and $\delta_{mm'}$ is the Kronecker delta.

The expansion functions chosen [2] that satisfy this orthogonality condition are given by equation (14):

$$\Gamma^{ijklg} = P_i(x) P_j(y) U_k(\mu) P_l(cos\ \phi) \delta(E - E_g). \tag{14}$$

In equation (14), Legendre polynomials are used to expand the spatial variables (x and y coordinates on a surface) and the azimuthal angle cosine ϕ. Chebyshev polynomials of the second kind are used to expand the polar angle cosine μ. A Dirac delta in energy E is equivalent to traditional multigroup energy treatment. It is evident that this expansion set satisfies isotropic angular flux for the 0th order angular expansion. Further, if the expansion coefficients $J_{s,m}^{i,+/-}$ are defined as

$$J_{s,m}^{i,+/-} = (\Gamma^m, \varphi_i^{+/-}), \tag{15}$$

then the 0th order expansion coefficients are simply the incoming and outgoing partial currents on mesh interfaces. Therefore, the partial currents are always preserved, satisfying the requirements on the expansion set.

The outgoing expansion coefficients for a coarse mesh can be computed by projecting the response functions onto the incoming expansion coefficients:

$$\sum_{s',m'} R^{i}_{ss',mm'} J^{i,-}_{s',m'} = J^{i,+}_{s,m}, \tag{16}$$

where $R^{i}_{ss',mm'}$ is the surface-to-surface response function, defined by

$$R^{i}_{ss',mm'} = (\Gamma_{m'}, R^{i}_{s,m}). \tag{17}$$

From this definition, it is seen that $R^{i}_{ss',mm'}$ is the outgoing flux of surface s in moment m in response to an incoming flux in moment m' on surface s'.

Global Problem and Solution

This formalism relies heavily on known response functions. These values are determined in a precalculation. While in a global problem quantities of interest must be computed for every region (including regions with redundant geometry and material composition), local response calculations only need to be carried out for unique coarse meshes, i.e., those with unique geometry and composition. While any method can be used to determine the response functions, COMET calculations employ the Monte Carlo method for calculations due to the method's ability to treat complex geometries.

During response generation, any quantities of interest can be computed. In practice, calculated values include surface-to-surface responses as well as the surface-to-volume responses for fuel pin fission density, neutron production, and neutron absorption. Figure 1 demonstrates the procedure through which responses are computed.

Figure 1. Assumed incoming fluxes shining onto surfaces (left) and outgoing fluxes and volume values in response (right) [16].

Assumed incoming fluxes of the form (14) are "shined," that is, applied as the boundary condition to equation (11) upon each surface of each unique mesh, and the responding quantities are tallied. This is equivalent to finding solutions to the boundary value problems (10-11) for each unique coarse mesh. Once all the boundary value problems are solved (all responses computed), the results are stored in a library.

After responses compiled in a library, a deterministic iterative procedure is used to compute the global solution to the problem. While exact solution methods vary by implementation [2,17], both inner and outer iterations are required to compute flux and eigenvalue solutions. The inner problem is defined by the eigenvalue problem

$$\sum_{s',m'} R^{i}_{ss',mm'}(k) J^{i,-}_{s',m'} = \lambda J^{i,+}_{s,m}. \tag{18}$$

The λ-eigenvalue in (18) has physical meaning. This eigenvalue represents the discontinuity between the incoming and outgoing partial currents in the global problem. When the response functions are evaluated at the true core eigenvalue, there is no discontinuity, and the λ-eigenvalue is equal to unity.

However, the true eigenvalue is not known before a calculation and must be determined iteratively. k is updated in outer iteration via the balance relation

$$k = \frac{\int d\boldsymbol{w}\boldsymbol{F}\psi}{L + \int d\boldsymbol{w}\boldsymbol{A}\psi}. \tag{19}$$

In equation (19), L is the system leakage, $\boldsymbol{w}$ denotes the space-angle-energy phase space, and $\boldsymbol{F}$ and $\boldsymbol{A}$ denote the fission and absorption operators, respectively. In order for a global solution to be computed, initial guesses for both flux and eigenvalue are made, and inner iterations on flux and outer iterations on eigenvalue are carried out until both are converged.

The exact solution methods used to solve the global problem have been the focus of much study. Of particular interest has been the acceleration of the solution method as well as appropriate handling of response data, which can be a formidable task when flux expansions are high and when many energy groups are used in computation of a solution. Details of these implementations will be discussed in the next chapter.

CHAPTER 3

CURRENT TECHNOLOGIES AND METHODOLOGIES

In this chapter, specific implementations of the solution of the inner and outer problems for deterministic COMET calculations will be discussed. Since COMET is a member of the response matrix method family, pertinent literature will be reviewed to discuss solution methods for this problem. In addition, current research efforts to accelerate COMET solutions will be discussed.

However, it had been found that these current acceleration methods only gain so much in terms of computational efficiency. This partly motivates the drive to extend COMET calculations to high performance computing applications. As such, current efforts at parallelization in modern transport codes will be discussed. In addition, the prevalent software implementations for developing parallel distributed and shared memory algorithms will be described.

3.1 Solution Methods for the Response Matrix Equations

As discussed in the previous chapter, the solution of the whole core solve in COMET calculations requires convergence on an inner problem (flux) and an outer problem (eigenvalue). Formally posed, the problem requires convergence on the principal eigenvector of the eigenvalue problem

$$MR(k)\boldsymbol{J}^{-} = \lambda \boldsymbol{J}^{-} \tag{20}$$

for each inner flux solve. In equation (20), $\boldsymbol{J}^{-}$ is the collection of all incoming expansion coefficients, $R(k)$ is the response matrix evaluated at global eigenvalue k, and M is the so-called connectivity matrix [17], the operator that transfers outgoing expansion coefficients to incoming expansion coefficients.

Once the inner eigenvalue problem has been solved, the updated vector $\boldsymbol{J}^{-}$ is used to update the global eigenvalue via the balance relation (19). In current COMET calculations, this balance is calculated via the equation

$$k = \frac{\sum_{i,s,,m} J_{s,m}^{i,-} RNF_{s,m}^{i}}{\sum_{i,s,,m} J_{s,m}^{i,-} RAB_{s,m}^{i} + \sum_{i \subset \partial V,s} (J_{s,0}^{i,-} - J_{s,0}^{i,+})} \tag{21}$$

where $RNF_{s,m}^{i}$ is the neutron production response for the i^{th} coarse mesh for surface s in moment m, $RAB_{s,m}^{i}$ is the neutron absorption response for the same conditions, and $\sum_{i \subset \partial V,s} (J_{s,0}^{i,-} - J_{s,0}^{i,+})$ is the net leakage out of the core.

Clearly, the inner problem is an eigenvalue problem itself, which requires an iterative method to obtain a solution. COMET calculations employ the power method [2] to solve this problem. The solution algorithm for both the inner and outer problem is outlined below.

1. **Initial Guess for Global Eigenvalue and Flux Distribution**
    a. A guess of unity for eigenvalue is often reasonable
    b. A guess of flat flux is used for any case
2. **Evaluate response functions at the global eigenvalue.**
    a. E.g., $R(k) = R(1)$
3. **Solve the inner problem given by equation (20) (Power method):**

a. Update outgoing expansion coefficients via equation (16), or in matrix form $R(k)\boldsymbol{J}^{n,-} = \boldsymbol{J}^{n+1,+}$

b. Update incoming expansion coefficients using interface and boundary conditions, i.e. apply the connectivity matrix: $M\boldsymbol{J}^{n+1,+} = \boldsymbol{J}^{n+1,-}$

c. Normalize expansion coefficients to outgoing partial current

d. If the outgoing partial currents are converged, then quit.

   i. Criteria: $\left|\frac{J_{s,0}^{i,+,n+1}}{J_{s,0}^{i,+,n}} - 1\right| < \varepsilon_{inner}$

4. **Update global eigenvalue via the balance relation given by equation (21).**
5. **Repeat steps (2-5) until both eigenvalue and expansion coefficients are converged.**

   i. Eigenvalue Criteria: $\left|\frac{k^{n+1}}{k^n} - 1\right| < \varepsilon_{outer}$

The power method has been employed in COMET calculations due to its robustness and stability in finding solutions for eigenvalue and principal eigenvector. However, as problem sizes grow large (e.g., for large reactors and/or problems with many energy groups and high flux expansions) the convergence rate for the power method in the inner problem can degrade significantly. Experience with the method has shown it may take as many as 1000s of iterations before convergence is reached.

3.2 Acceleration Schemes for COMET Calculations

While other implementations of response matrix methods [17] have used Newton's method [18] to solve the inner eigenvalue problem, the power method is the selected solution method for implementation in this study. The power method is selected due to its cheap computational cost at every iteration, robust nature in converging to the principal eigenvector, and is amenable to the current organization of the response library in the code, which pre-builds submatrices for use in the deterministic solve. However, the problem of potentially slow convergence in the inner problem persists. Zhang and Rahnema [2] have suggested two acceleration methods – Low Order Acceleration and Chebyshev Polynomial Filtering – that have been observed to work well for reactor calculations. These schemes are described below.

Low Order Acceleration

In the description of the power method in the previous subsection, initial guesses for core eigenvalue and flux were discussed. As previously stated, it is common for initial guesses to be unity for k and flat for $\boldsymbol{J}^{-}$. However, a first guess can be improved if the result from a lower-order calculation is used as initial data. This is the idea behind Low Order Acceleration (LOA). In a deterministic core calculation, a low order calculation is carried out. Current implementations carry out a second-order calculation in space and angle. While this solution may not give desired accuracy alone, they serve as a much improved initial guess for core eigenvalue and principal eigenvector $\boldsymbol{J}^{-}$. Formally, LOA is written

$$k^{H}(1) = k^{L}(U), \tag{22}$$

$$J_{s,m}^{i,-,H}(1,1) = \begin{cases} J_{s,m}^{i,-,L} \; for \; \mathrm{s} = 1, \dots, M^{L} \\ 0 \text{ for s} = \; M^{L+1}, \dots, M^{H} \end{cases}. \tag{23}$$

In equations (22-23), the superscripts "H" and "L" represent high-order and low-order calculations, respectively. U is the low-order converged iterate for k. An initial guess using this low-ordered result has been seen to improve computational performance by a factor of 2-3 [2]. In addition, this author's experience suggests that LOA provides a good first-blush view of reactor eigenvalue and power distribution, even if a high-ordered answer is desired.

Chebyshev Polynomial Filtering

In addition to LOA, Chebyshev polynomial filtering has been used to accelerate convergence on the inner problem. For problems with dominance ratio near unity, power iterations can have very low convergence rate. This convergence can be improved by breaking each iteration into sub-iterations based upon polynomials of the initial matrix. The forms of the polynomials p_q are given below:

$$\sum_{s',m'} p_q(R_{ss',mm'}^{i}) J_{s',m'}^{i,-,(n)} = J_{s,m}^{i,+,(n+1)}, \tag{24}$$

$$p_q(R) = \prod_{i=1}^{q} (R - \; w_i I), \tag{25}$$

$$w_i = \gamma \left[\cos\left(\frac{2i-1}{2q}\right)\pi + 1 \right] - \; \alpha. \tag{26}$$

In equations (24-26), w_i is the weighting function used for over/under relaxation depending upon the problem solved. i is the sub-iteration index. γ, q, and α are potentially problem-dependent parameters that are determined empirically. Zhang and Rahnema echo Stamm'ler and Abbate [19] in saying that the values 0.985, 10, and 1.0, respectively, work well for problems studied. In the work of this

thesis, the values 0.99, 10, and 1.01 have been used and have been seen to work well.

It should be noted that choice of parameters can severely impact convergence behavior. While these selected parameters have certainly shown to improve performance – Zhang and Rahnema [2] claim that these parameters used in Chebyshev Polynomial Filtering, combined with LOA, have improved computational efficiency by at least a factor of 20 – poorly chosen parameters can severely degrade convergence. While not observed directly in this study, the problem-dependent nature of these parameters calls for a more systematic selection. Some preliminary work of this thesis toggled these parameters experimentally, but a true systematic selection of these parameters would require analysis on the eigenspectrum of the $MR(k)$ matrix, which was outside the scope of this work. A fuller analysis of this problem should prove to be an interesting area of further study.

Ongoing Work in Improving Efficiency

An ongoing research project with the goal of improving efficiency of the COMET method is adaptive COMET. Traditional COMET implementations use a flux expansion and hold this expansion constant throughout a problem. In contrast, a method was developed by Remley and Rahnema [6-9] that allows the flux expansion to vary depending upon problem-dependent factors. The gains in efficiency from using this adaptive method over a standard COMET method varied from speedups of 2-5.8 [8-9].

3.3 Parallel Computing in Transport Calculations

Prior to the work of this study, parallel computing has been used extensively by the community to improve computational efficiency in transport calculations and extend the range of problems that can be solved with existing methods. While computational efficiency and accuracy of calculations improve with continuing research in advanced methods for solving the neutron transport equation, it is recognized that present solution methods are greatly improved by developing software implementations that take advantage of modern computing architectures, allowing for multiprocessing and advancements such as utilizing gpu systems. Before describing previous work of parallel computing in transport codes, it is useful to discuss common concepts germane to multiprocessing.

Basic Concepts of Parallel Computing

The first important concept in parallel computing is the type of computer architecture being used in development of applications. Classically, four types of machines have been described [20]:

SISD – Single Instruction/Single Data Stream
SIMD – Single Instruction/Multiple Data Stream
MISD – Multiple Instruction/Single Data Stream
MIMD – Multiple Instruction/Multiple Data Stream

SISD systems describe computers where applications are applied sequentially; one instruction is carried out before the next one begins. SIMD machines apply the same operations to multiple data sets. MISD is a rarely used classification for machines. MIMD is the more commonly used machines used in parallel computing applications.

MIMD machines come in two flavors: shared and distributed memory. Shared memory systems allow each processor to have access to a global memory. These systems are typically limited to tens of processors due to connections needs to the memory map. Distributed memory systems offer completely independent memory between nodes. A node is a collection of processors with potentially shared memory. Information that is shared between processes on different nodes must be communicated via networks. There is virtually no limit to the number of processors on a distributed memory system; machines range from one to thousands of nodes.

In assessing applications utilizing parallel computing, accuracy of calculations is insufficient; computational performance is also important in evaluating programs. To this end, it is important to define *parallel speedup* and *efficiency*, given below

$$Speedup = \frac{T_S}{T_N}, \tag{27}$$

$$Efficiency = \frac{Speedup}{N}. \tag{28}$$

In equations (27-28), T_S is the serial execution time of a parallel program, and T_N is the parallel execution time on N processors. Efficiency is the speedup normalized to the number of processors used in an application and is a measure of the economy of parallelization. It should be noted that a common assumption [21] (one utilized in this study) is that the overhead in parallelizing a program is negligible when executing a program on a single processor. That is, the serial execution time in analysis is that of a "parallelized" code executed on a single processor, rather than

older version of a code that was implemented in serial without considerations for parallelization.

It is tempting to believe that utilizing more processors will always result in increased speedup. However, parallel programs have an upper bound in speedup achievable limited by their parallel fraction f_p. This upper bound on speedup is determined by *Amdahl's Law*:

$$S_p = \frac{1}{(1-f_p)+ {f_p}/{N}}. \tag{29}$$

In equation (29), S_p is the maximum theoretical speedup achievable for a parallel program given a parallel fraction and number of processors used. As N tends to infinity, the maximum theoretical speedup is limited to the inverse of $1 - f_p$. From this, one sees that a high parallel fraction is of utmost importance if high parallel speedups are desired. In practice, other considerations, such as communication costs, computer architecture, or the problem solved, adding more and more processors to a computation will greatly impact parallel efficiency.

Amdahl's Law leads programmers to consider the parallelizability of various algorithms and programs. Although parallelizability itself is a qualitative term that might differ between communities, it is useful to distinguish between different types of algorithms and their amenability to parallelization. If a program or algorithm requires communication or synchronization many times per second or after unit numbers of operations, then the level of parallelism is known as *fine-grain parallelism*. If processes do not need to communicate many times per second or if many operations are carried out between communications, then the level of parallelism is known as *coarse-grain parallelism*. If processes never have

to communicate, then we say these processes are *embarrassingly parallel*. An example of an embarrassingly parallel process is response generation for COMET calculations. Unique responses can and are generated completely in parallel without the need for any synchronization.

The discussion of parallelizability brings up the concept of *granularity*, which is a measure of the number of operations that are carried out in between synchronization or communication. Since communication or synchronization are serial elements in a program that can limit performance, it is generally seen that a greater grain size of an application will lead to better parallel performance.

In practice, parallel efficiency is tied not only to parallel fraction of a program but also the costs of synchronization or communication. Therefore, an important aspect of analysis of parallel applications is the cost of these synchronizations and communications. For high parallel scalability (the ability to apply a parallel program efficiently to many processors), typically computation costs should dominate communication costs. This is even more important when one keeps in mind that communication speed is often much smaller than computation speed on modern computing architectures [22].

Software Implementations

When developing codes to run in parallel, one must make the jump from theoretical considerations to practical applications. The most important distinction one must make is the software implementation to be used in parallelization. While many software implementations have been developed [23],

two main software implementations have been used in parallelization: OpenMP and MPI [22].

OpenMP allows for parallelism based upon shared memory MIMD systems. With OpenMP, multiple *threads*, or tasks, are created and are executed simultaneously. Because this parallelism occurs in a shared memory environment, communication between tasks is not required. This implementation is useful for situations where there is *data parallelism*, when the same calculation is carried out on the same or multiple sets of data. OpenMP has the advantage of simple implementation, as parallelism is determined from a small set of compiler directives, and incremental parallelism of a code without the need for large rewrites. However, a significant limitation to OpenMP is the requirement for shared memory. Because of this limitation, the number of processors one can use in an OpenMP calculation is limited to the number of processors that have access to a global memory.

Message Passing Interface, or MPI, is another prevalent software implementation for parallel programs. In contrast to OpenMP, MPI is based upon the distributed memory MIMD model, so communication between processes is required. MPI is useful for situations where there is *task parallelism*, or when different tasks are carried out on the same or different sets of data. MPI does carry with it the requirement that typically a large amount of code rewriting is necessary when developing a parallel program, and care must be made that distributed algorithm is designed and implemented such that optimal computational performance is achieved. A significant advantage of MPI is that MPI is not limited

to any number of processors like OpenMP is; indeed, distributed algorithms designed and implemented with MPI are more generally applicable and portable than parallel programs with OpenMP. Because of its advantages, MPI has been widely implemented in transport codes (including the ones discussed here) and is the focus of software implementation in this study.

Transport Codes That Employ Parallelism

Both stochastic and deterministic codes have utilized parallel computing in their software applications to great success. In addition, codes employing advanced methods have also have parallel implementations. Perhaps the most amenable methods to parallelization are Monte Carlo methods. These calculations have a very coarse granularity and thus can achieve high parallel efficiencies. Codes such as MCNP [24] have seen significant speedups from the use of parallel software implementation. More recently, a Monte Carlo response generation code SPaRC has been developed that implements parallelism via MPI, and that work saw orders of magnitude speedup [11].

Parallelism has also been applied to deterministic methods. The Parallel Environment Neutral-particle TRANsport (PENTRAN) code developed by Sjoden and Haghighat [21] employs finite-difference discrete ordinates multigroup transport distributed across many cores. Scalable in memory, the PENTRAN code is able to solve larger problems than can be solved on a serial machine. With a parallel fraction as high as 0.98, speedups as high as 50 have been observed [21].

The response matrix code Serment [25] has also implemented parallelism. Emphasis on the parallelism in this code is on the response generation and solution

of the nonlinear response matrix problem, which is facilitated with PETSc and SLEPc libraries [25]. However, the code has yet to be applied to full core problems on par with previously developed response matrix codes such as COMET. The use of parallel computing for efficient whole-core solves in a response matrix method is a novel development and the principal goal of this study.

CHAPTER 4

DEVELOPMENT OF THE DISTRIBUTED ALGORITHM

The main focus of this study was the development and implementation of a distributed (local memory only) algorithm for COMET calculations. The distributed memory approach was chosen because of its ability to explicitly to express parallel algorithms, and it is generalizable many different computing systems. To this end, the primary software implementation used in this study was MPI. In this chapter, the distributed algorithm is described.

4.1 Problem Decomposition

In order to parallelize a calculation, a problem must be decomposed in some fashion. Different "chunks" of the problem, including relevant data, are allocated to different processors, and each process works only on the chunk to which it is assigned. Communication occurs as necessary in the computation. It is well known that communication costs are a major bottleneck in computations [21-22], so an algorithm should be developed to limit these potential degradations in performance.

COMET calculations offer multiple ways in which problems may be decomposed for solution on many processors. The fundamental observation is that, during the iteration, the outgoing expansion coefficients are computed via equation (16). During these calculations, there are no dependencies on other parts

of the algorithm. These calculations entail the majority of computational effort in COMET calculations, and therefore, the algorithm is amenable to parallelization. However, computational performance depends upon how these computations are decomposed onto multiple processors. Three possible decompositions include decomposition by surface, decomposition by flux expansion, and decomposition by domain.

Decomposition by Surface

To describe decomposition by surface, it is instructive to construct the response matrices used in COMET calculations from the terms in equation (16). While these updates in outgoing expansion coefficients compute expansion coefficients for a given surface *and* moment, it is more efficient to update the expansion coefficients for a given surface at *every* moment in a matrix-vector multiplication, that is,

$$\boldsymbol{J}_s^{i,+} = \sum_{s'} R_{ss'}^i \boldsymbol{J}_{s'}^{i,-}. \tag{30}$$

With these matrix-vector multiplications, the expansion vectors and response submatrices are defined as

$$J_s^{i,+/-} = \begin{pmatrix} J_{s,1}^{i,+/-} \\ \vdots \\ J_{s,m}^{i,+/-} \\ \vdots \\ J_{s,M}^{i,+/-} \end{pmatrix}, \tag{31}$$

$$R_{ss'}^i = \begin{pmatrix} R_{ss',11}^i & \cdots & R_{ss',m1}^i \\ \vdots & \ddots & \vdots \\ R_{ss',1m'}^i & \cdots & R_{ss',mm'}^i \end{pmatrix}. \tag{32}$$

With updates in expansion coefficients carried out in this fashion, it is possible to partition the calculation along surfaces. Each update of the form of equation (30) can be partitioned onto up to s processors. However, this decomposition is extremely limited, and communication between processors is still required when updating incoming expansion coefficients from outgoing expansion coefficients. The calculation can be decomposed further by distributing the individual matrix vector products $R_{ss'}^{i}\boldsymbol{J}_{s'}^{i,-}$ onto different processors, which would allow for a maximum decomposition of up to s^2 processors, which would allow for decomposition onto $O(10)$ processors. For example, for the Cartesian geometry COMET code off of which this work is based, this would lead to a maximum number of processors of 36. While this is a larger decomposition than simply partitioning along surfaces, another set of communications is required, as s matrix-vector products are summed for a single update in $\boldsymbol{J}_{s}^{i,+}$.

Decomposition by Flux Expansion

Decomposition by flux expansion involves splitting the submatrices $R_{ss'}^{i}$ into rectangular chunks that are separately multiplied by incoming expansion coefficient vectors $\boldsymbol{J}_{s'}^{i,-}$. This decomposition is demonstrated in the figure below.

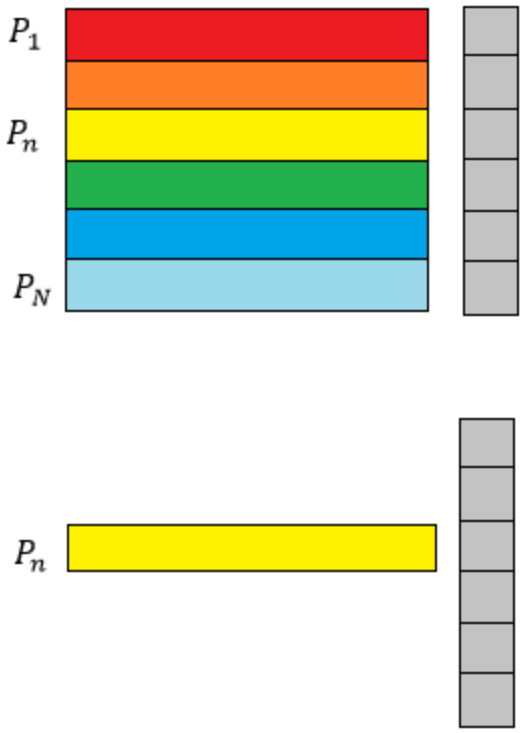

Figure 2. A matrix (top left) is decomposed into rectangular elements (bottom left) whose matrix-vector products are evaluated in parallel.

This decomposition has the advantage of being bounded by the submatrix dimension *sm*, which for large problems can be in the hundreds. However, care must be made to not decompose too finely, since the granularity of this calculation can become so fine that the matrix-vector evaluation will not scale well. An example of this scaling issue can found in reference [22]. These types of decompositions would likely be similar to the surface decomposition as a decomposition onto $O(10)$ processors as a theoretical maximum.

Decomposition by Domain

Finally, a proposed decomposition is by domain. A reactor problem is broken into different regions in space and given to different processors to perform

calculations independently. Domain decomposition has the intuitive advantage that this is the method that is used to solve COMET problems already; local solutions are computed and the global solution is constructed from these results.

Domain decomposition also has mathematical justification. The structure of the response matrix used in the update $R(k)\boldsymbol{J}^{n,-} = \boldsymbol{J}^{n+1,+}$ of the power method solution is block diagonal:

$$R(k) = \begin{pmatrix} R_1(k) & & & & & \\ & \ddots & & & & \\ & & R_i(k) & & & \\ & & & R_{i+1}(k) & & \\ & & & & \ddots & \\ & & & & & R_N(k) \end{pmatrix}, \tag{33}$$

where N is the total number of coarse meshes in a problem. The block matrices $R_i(k)$ are given by

$$R_i(k) = \begin{pmatrix} R_{11}^i & \cdots & R_{s1}^i \\ \vdots & \ddots & \vdots \\ R_{1s'}^i & \cdots & R_{ss'}^i \end{pmatrix}. \tag{34}$$

This structure allows expansion coefficients for each coarse mesh to be computed independently in a sweep via

$$\boldsymbol{J}^{i,+} = R_i(k)\boldsymbol{J}^{i,-}, \tag{35}$$

where

$$\boldsymbol{J}^{i,+/-} = \begin{pmatrix} \boldsymbol{J}_1^{i,+/-} \\ \vdots \\ \boldsymbol{J}_s^{i,+/-} \\ \vdots \\ \boldsymbol{J}_S^{i,+/-} \end{pmatrix}. \tag{36}$$

A significant advantage of this decomposition is that it is limited only by the number of coarse meshes used in a problem. For full reactor problems, there are

thousands of coarse meshes used. Unlike the decomposition of flux expansion proposed in the previous subsection, each update of the form (35) requires many calculations, ensuring synchronization or communication is not required prohibitively often; the granularity of a calculation is coarse enough to allow for decomposition onto many processors. Because of this, domain decomposition is seen as having the theoretical possibility for decomposition onto $O(1000)$ processors. Due to the much higher limit of parallelization that domain decomposition offers over the other proposed decompositions discussed, it is the method of decomposition used in the distributed algorithm.

Domain decomposition allows for a splitting of the calculation, but transfers of outgoing expansion coefficients to incoming coefficients during the power iteration still require communication between processors. If an outgoing expansion coefficient on a given processor transfers to an incoming expansion coefficient on an adjacent coarse mesh that exists on a different processor, then these processors must communicate. Therefore, the decomposition that takes place in calculations should allow for meshes to be spatially close together to minimize this need for synchronization at every iteration. Figure 3 demonstrates how a problem is decomposed.

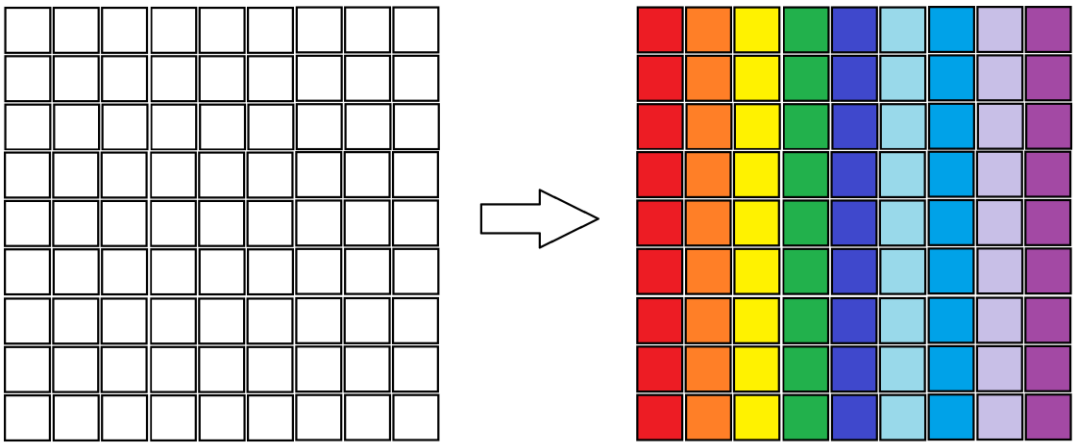

Figure 3. Decomposition of a domain (left) to multiple processors (right), with a different color representing the part of a domain on a particular processor.

Of course, the decomposition shown in Figure 3 is the case of an even decomposition; that is, when the number of coarse meshes in a problem and the number of processors used in the problem solution divide without remainder. However, for many applications of the algorithm, the decomposition will lead to partitions on different processors that are *not* even, since in general the number of coarse meshes and processors will not divide evenly. This brings up the issue of *load balancing*. The processors employed to perform a COMET calculation should do similar amounts of work, otherwise some processors will sit idle for lengths of time during a calculation, introducing inefficiencies.

The decomposition used in the distributed algorithm addresses load balancing by enforcing the following rule when assigning coarse meshes to various processors:

$$N_{CM} = \begin{cases} floor\left(\frac{CM}{N}\right) + 1, i < mod(CM, N) \\ floor\left(\frac{CM}{N}\right), otherwise \end{cases}. \tag{37}$$

In the rule (37), CM is the number of coarse meshes in a problem, N_{CM} is the number of coarse meshes on a processor, and i is the coarse mesh index. Because of this rule, computational load between processors is relatively balanced; the worst case scenario is some processors having one more coarse mesh than other processors. In the limiting case where there are many coarse meshes distributed to each processor in a calculation, this imbalance becomes small. Domain decomposition for the uneven case is demonstrated in Figure 4.

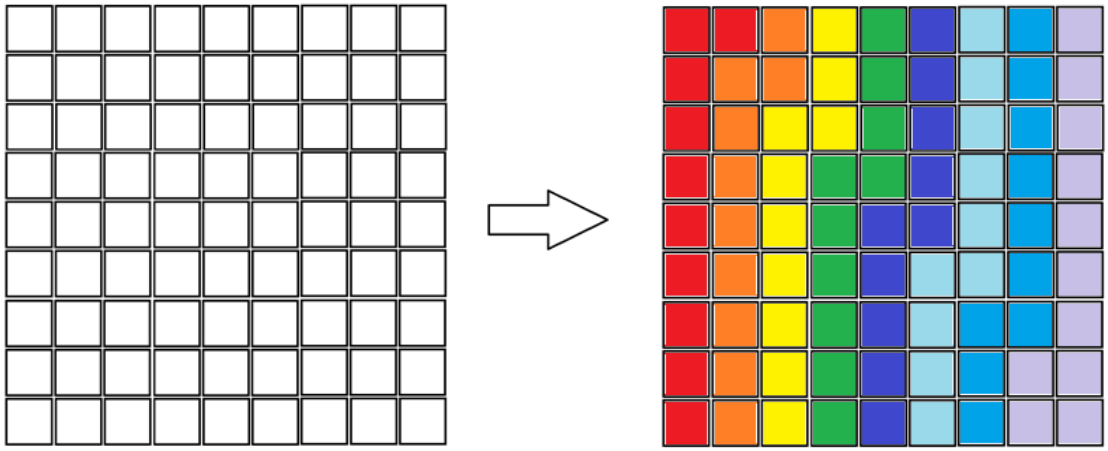

Figure 4. Domain decomposition for COMET calculations in the uneven case.

It should be noted that domain decomposition occurs axially first. That is, coarse meshes are "filled up," or selected, along every axial column first before moving to another (x,y) location. Domain decomposition occurs one axial column at a time. Other forms of domain decomposition, such as breaking the problem into cubes that are solved on different processors, are possible as well. These alternative decompositions might even further limit the communication costs at every iteration. However, it is more difficult to systematically and automatically decompose the domain in this way, especially for the uneven decomposition case.

More sophisticated domain decomposition methods can be investigated as a part of future work.

4.2 Distinctions from the Serial Algorithm

Aside from the decomposition of the problem, other changes to the solution algorithm described in the previous chapter have been made, either taking advantage of parallel processing or facilitating it.

Reading in Responses

In the execution of the COMET code, responses are read from a stored response library and evaluated at the current guess for global eigenvalue. In the serial implementation of the solution algorithm, all responses were read from the library regardless of their appearance in the problem. This has been modified in the distributed algorithm. Once domain decomposition has occurred, each subdomain is surveyed for the response data it needs to perform its coarse mesh calculations. Each process then only reads the required response data from the database. This modification is demonstrated in Figure 5.

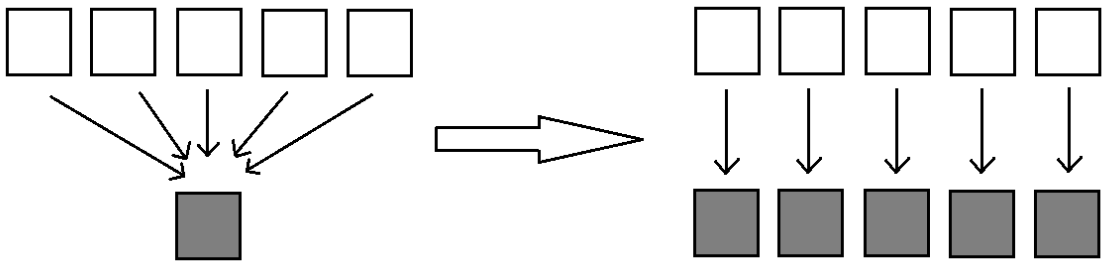

Figure 5. Modification in data access model from the serial (left) to the distributed (right) algorithm.

In Figure 5, the white blocks represent response data, and the grey blocks represent processors that are carrying out all or part of COMET calculations. This modification allows for memory resources on each processor to be dedicated only for relevant response data, which can be a precious resource when response libraries grow large.

The modification is also a step in the right direction for coupling COMET calculations to on-the-fly response generation, which is an interest of future work. In this case, response data will exist in a distributed environment rather than at a single location (in this case a CDF database [26]). Once a decomposition has been made, the relevant response data can be sent only to the processors that need it for calculations. If on-the-fly response generation is coupled to a serial implementation of the COMET code, response data must be collapsed to a single location before being used in further COMET calculations. The difference in data flow between serial and distributed COMET calculations coupled with on-the-fly response generation is demonstrating pictorially in Figure 6.

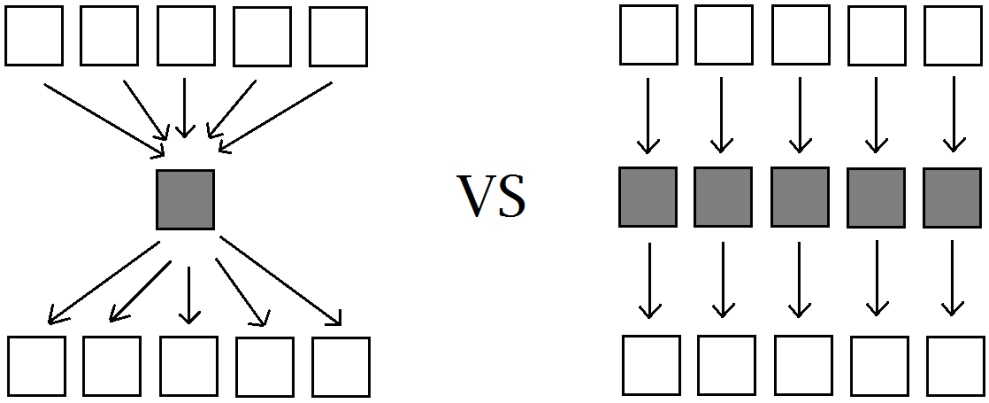

Figure 6. On-the-fly response generation coupled with serial (left) vs. distributed (right) implementations of the COMET code.

In Figure 6, as in Figure 5, the white blocks represent response data, and the grey blocks represent processors carrying out COMET calculations. A distributed implementation of the COMET code allows for more streamlined data transfer between response generation and COMET calculations. For a further discussion of the potential of coupling on-the-fly response generation with COMET calculations (as well as multiphysics), the reader is encouraged to consult the references [10-11].

Sweep Order

Sweep order, the order in which coarse mesh calculations (35) are carried out, has also been changed in the distributed algorithm from the serial version. In practice, the sweep order employed is arbitrary, since the incoming expansion coefficients used in the calculation are from the previous iteration (this is similar in spirit to a Jacobi iteration when solving systems of linear equations).

In the serial algorithm, the sweep order was determined on a *material* (response function) basis. This was to limit the look-up cost of response submatrices $R^{i}_{ss'}$. For a given unique coarse mesh type, the corresponding response submatrix was fetched from memory, and calculations using this response data were carried out for *all* repetitions of the unique coarse mesh regardless of where they occur in a problem.

In the distributed algorithm, the sweep order is determined on a *geometric* basis. This distinction from the serial algorithm is demonstrated in Figure 7.

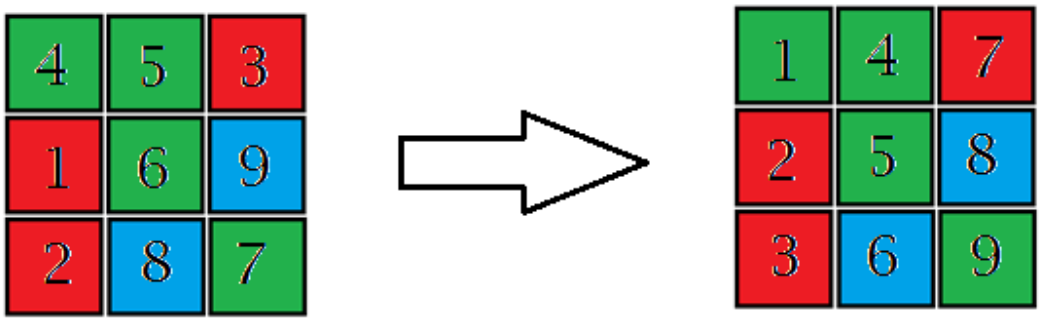

Figure 7. Sweep orders for the serial (left) vs. the distributed (right) algorithm.

In Figure 7, each block represents a coarse mesh, and each color represents a unique coarse mesh, necessitating new response data. The numbers on each block represent the order in which the coarse mesh calculation (35) was carried out. The change in sweep order was made because it is generalizable to any number of processors used in a calculation – the sweep can be performed along the domain decomposition without regard for differences in coarse meshes on different processors. The change in sweep order also anticipates future problems that the code will encounter. For realistic reactor problems, the number of unique

coarse meshes approaches the number of total coarse meshes in a problem due to the heterogeneity effects of factors such as burnup. For cases such as these, material sweeping becomes less viable.

Communication at Each Iteration

As stated previously, the stage of the iteration that transfers outgoing expansion coefficients to incoming expansion coefficients requires communication between processors. This communication only occurs on the boundaries between subdomains that exist on different processors to minimize the cost of this step of the algorithm. Communications that occur at every iteration are demonstrated in Figure 8.

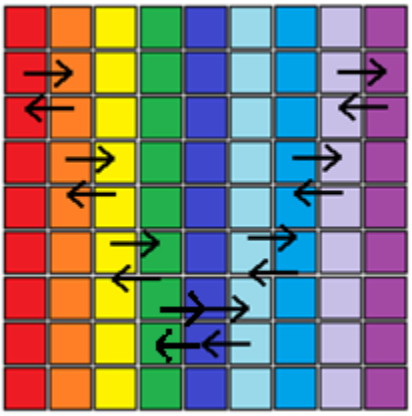

Figure 8. Communication pattern used at each iteration.

As in previous figures, the blocks in Figure 8 represent coarse meshes, and the different colors represent coarse meshes on different processors. The arrows represent a surface where inter-processor communication takes place.

Once the problem has been decomposed and the processors employed in a calculation have been assigned their respective coarse meshes, the addresses for where outgoing expansion coefficients are to be sent are determined. These

addresses are used during the communication at each iteration in conjunction with the MPI_SEND and MPI_RECV commands, which are the software implementations to send specific messages between processors in MPI. In the communications, all needed expansion coefficients are sent out via MPI_SEND, and then all expansion coefficients are received. This is done to avoid locking in the program execution; if a processor is expecting to receive some data via MPI_RECV but it has not been sent by another processor via MPI_SEND (due to different sending priorities because of the coarse mesh loading on different computers), execution of the program is halted.

Calculation of Balance Values

The primary focus thus far on the distributed algorithm's distinctions from the serial algorithm has been on the inner iteration. This is sensible; the inner problem is the majority of the computational expense in COMET calculations, as the outer problem simply updates eigenvalue via the balance relation (21). However, when performing a calculation where coarse mesh data is distributed across many processors, consideration of calculating these balance values (neutron production, neutron absorption, and leakage) must be made; the coarse mesh calculations in the inner problem can be carried out simultaneously, but balance values are single numbers that are best calculated on a single processor. To compute these values, each processor determines its own volume responses for production, absorption, and leakage. Global values are calculated by *reducing* these values with the command MPI_REDUCE. Global eigenvalue is calculated via

(21) on a single processor and then sent to all processes with MPI_BCAST for the purposes of evaluating responses.

4.3 Outlined Distributed Algorithm

While the decomposition and special considerations in developing the distributed algorithm have been made, it is instructive to outline the algorithm, which highlights the changes that have been made from the serial algorithm. The distributed algorithm is given below.

1. **Given a number of coarse meshes and number of processors, decompose the problem**
    a. Even division case: decompose as in Figure 3
    b. Uneven division case: decompose as in Figure 4
    c. Determine the addresses for all expansion coefficients that need to be communicated at each inner iteration
2. **Initial guess for global eigenvalue and flux distribution**
    a. Eigenvalue sent to all processors via MPI_BCAST
3. **Evaluate response functions at the global eigenvalue.**
    a. Processors only read in relevant response data
    b. E.g., $R(k) = R(1)$
4. **Solve the inner problem – equation (20) (Power method):**
    a. Update outgoing expansion coefficients via equation (16), sweeping geometrically rather than by material, as in Figure 7

b. Update incoming expansion coefficients using interface and boundary conditions, i.e. apply the connectivity matrix: $M\boldsymbol{J}^{n+1,+} = \boldsymbol{J}^{n+1,-}$

   i. If incoming expansion coefficients cross processors, they must communicate

   ii. Send out all needed expansion coefficients via MPI_SEND

   iii. Receive all needed expansion coefficients via MPI_RECV

      1. Sends and receives sent to addresses determined in step 1 of the algorithm

c. Normalize expansion coefficients to outgoing partial current

d. If the outgoing partial currents are converged, then quit.

5. Update global eigenvalue via the balance relation – equation (21).

a. Each processor computes its relevant balance values (production, absorption, leakage)

b. Sum over all processors for balance values via MPI_REDUCE

c. Eigenvalue computed on a single processor.

6. Repeat steps (2-5) until both eigenvalue and expansion coefficients are converged.

4.4 Scalability Analysis

Once a distributed algorithm has been proposed, it is important to assess its scalability. In a simple sense, this is analyzing the ratio cost of communications (which can limit the efficiency of a parallel program) to the cost of computations.

For an algorithm to be scalable, this ratio of communication to calculation should be as small as possible. Further, it is desirable that for larger problems – those with more computations required – that scalability will increase, that is, the ratio of communication to computation will decrease.

To determine if this is the case for the developed distributed COMET algorithm, let some assumptions be stated. First, this analysis only applies to the inner iterations – the bulk of the computational effort in COMET calculations. Second, assume that calculation and communication have uniform costs per operation, called T_{calc} and T_{comm}, respectively. With this assumption, calculation and communication costs can be estimated as integer multiples of these values. While these types of analyses may not be exact, they can give insight into how the algorithm is expected to behave when encountering different types of problems.

In the communication step of the inner iteration, each expansion coefficient vector that must be sent between processors as size m, the size of the flux expansion used in the problem. When one expansion coefficient vector is sent from a processor to another, it must also receive a corresponding expansion coefficient vector. Therefore, every instance of communication requires $2mT_{comm}$. Communication is required on every surface that is shared by coarse meshes that exist on different processors, so let the total communication cost at an iteration be given as $2amT_{comm}$, where a is the number of surfaces where communication must take place. a can be difficult to estimate, as it depends upon the topology of the decomposition. However, for the limiting cases where there are many coarse meshes on a processor with relatively few surfaces that require communications

($N \ll CM$) and the cases where there are few coarse meshes on a processor and nearly every surface requires communications ($N \approx CM$), a can be estimated as

$$a \approx \begin{cases} 1, N \ll CM \\ sCM, N \approx CM \end{cases}. \tag{38}$$

Therefore, reasonable estimates for communication costs in the limiting cases are

$$Cost_{comm} \approx \begin{cases} 2mT_{comm}, N \ll CM \\ 2sCMmT_{comm}, N \approx CM \end{cases}. \tag{39}$$

While not exact, these estimates demonstrate that as the number of processors used in a calculation is increased while number of coarse meshes is held constant, communication costs increase, which could impact the scalability of the algorithm.

To assess the costs of calculation in an iteration, it is useful to describe each computational task and its associated expected number of operations, keeping in mind the total cost accrued over all operations. Each iteration involves the computation of the matrix-vector product $R_{ss'}^{i}\boldsymbol{J}_{s'}^{i,-}$, which costs $2m^2 - m$ floating point operations. For each $\boldsymbol{J}_{s}^{i,+}$to be computed, this calculation happens s times; there is the additional cost of s^2m additions, so the running total of operations is $s^2m + s(2m^2 - m)$. The computation of $\boldsymbol{J}^{i,+}$is s times the calculation cost of $\boldsymbol{J}_{s}^{i,+}$. The number of operations is now $s^3m + s^2(2m^2 - m)$. To determine $\boldsymbol{J}^{i,-}$, the connectivity matrix must be applied to the outgoing expansion coefficients as $M\boldsymbol{J}^{n+1,+} = \boldsymbol{J}^{n+1,-}$. The connectivity matrix is sparse; there is at most one nonzero element in each row. Therefore, this matrix-vector multiplication only introduces an additional calculation cost of sm. This cost is multiplied by the number of coarse meshes to obtain the total estimated calculation cost:

$$Cost_{calc} = CM(s^3m + s^2(2m^2 - m) + sm)T_{calc}\,. \tag{40}$$

The ratio of communication to computation is then

$$\frac{2T_{comm}}{CM(s^3+s^2(2m\ -)+s)T_{calc}} \tag{41}$$

in the limiting case for $N \ll CM$. In the limiting case for $N \approx CM$, the ratio of communication to computation times is

$$\frac{2T_{comm}}{(s^2+s\ \ (2m\ -1)+1)T_{calc}}. \tag{42}$$

In both limiting cases, it is seen that calculation cost dominates communication cost. Further, as problem sizes grow larger by increasing number of coarse meshes or by increasing the flux expansion (growth in m), the scalability of the algorithm is expected to increase.

This model of scalability is encouraging. However, it is desirable to observe how the algorithm behaves in practice. Chapter 6 focuses on this issue by performing a sensitivity study of the effects of problem size (both in terms of number of coarse meshes as well as flux expansion) on efficiency of the distributed algorithm implemented in the COMET code.

CHAPTER 5

ANALYSIS OF THE COMET-MPI CODE WITH AMDAHL'S LAW

In order to assess the performance of the distributed algorithm in the COMET code, an implementation was developed, which will henceforth be referred to as COMET-MPI, a *proxy app* [23] to the main COMET *research app*. COMET-MPI retains much of the functionality of the main COMET code, but it has been modified to allow for parallelization with MPI. For instance, the response libraries are still stored in and accessed from CDF databases [27] while the deterministic whole-core solutions are carried out by the algorithms described previously. The code has been optimized to deal with PWR-type benchmark problems with symmetry for the purpose of focusing on parallel performance of the code; these optimizations do not limit the method.

To assess the performance of the COMET-MPI code, a computational model for the C5G7 problem [28] was developed. COMET-MPI solved this problem for various number of processors, and the important aspects of computational performance were determined and analyzed. Different aspects of the algorithm were isolated to determine their effect on parallel efficiency and computational performance. As expected, two primary factors in performance were the calculation effort such as determination of the response matrix-vector products and the communication cost between each processor during each iteration.

However, another factor that proved to greatly impact parallel performance was the need to communicate with memory in the form of looking up response functions during each coarse mesh sweep.

It is important to demonstrate that the behavior of a developed code, particularly for a novel development like COMET-MPI, which offers a unique consideration of communication with memory, matches well with theory. As such, the performance data were compared against a model using Amdahl's Law (29) to determine whether the code performed as expected. Further, Amdahl's Law was used to draw conclusions on the parallelizability of the code as well as potential bottlenecks. For a thorough description of the problem solved, the reader is encouraged to review Chapter 6, which describes the benchmark in detail.

5.1 Effect of Response Function Lookups on Performance

When using the COMET-MPI code to solve the C5G7 problem, it was determined that the cost of response function lookups (to be described shortly) greatly impacted computational performance of the code. To demonstrate this effect, various computational performance data are presented. In all problem cases, the C5G7 problem was solved with a fixed number of iterations: 1 outer iteration to evaluate the response functions and 150 inner iterations to converge the problem. No acceleration techniques were used. Because much of the effort in the algorithm and code focuses on the inner iteration, performance data for the inner iterations alone were recorded. To ensure the data were reliable, all cases

were run on a dedicated homogeneous research cluster of Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz processors, with 2 nodes of 20 processors each.

The wall-clock times for COMET-MPI solutions to the C5G7 problem under these conditions were determined for two cases. The first case was a problem solution with response function lookups, which shows the effect on computational performance of both inter-processor communication as well as communication with memory. The second case was a problem solution without response function lookups, which only shows the effect of inter-processor communication on computational performance. While response function lookups are a vital part of the solution algorithm, the focus of this study was to assess factors of computational performance. Figure 9 gives the wall-clock times for COMET-MPI solutions for these two cases for 1-40 processors. For both cases, Figure 10 gives the speedups from parallelization, and Figure 11 gives the parallel efficiencies.

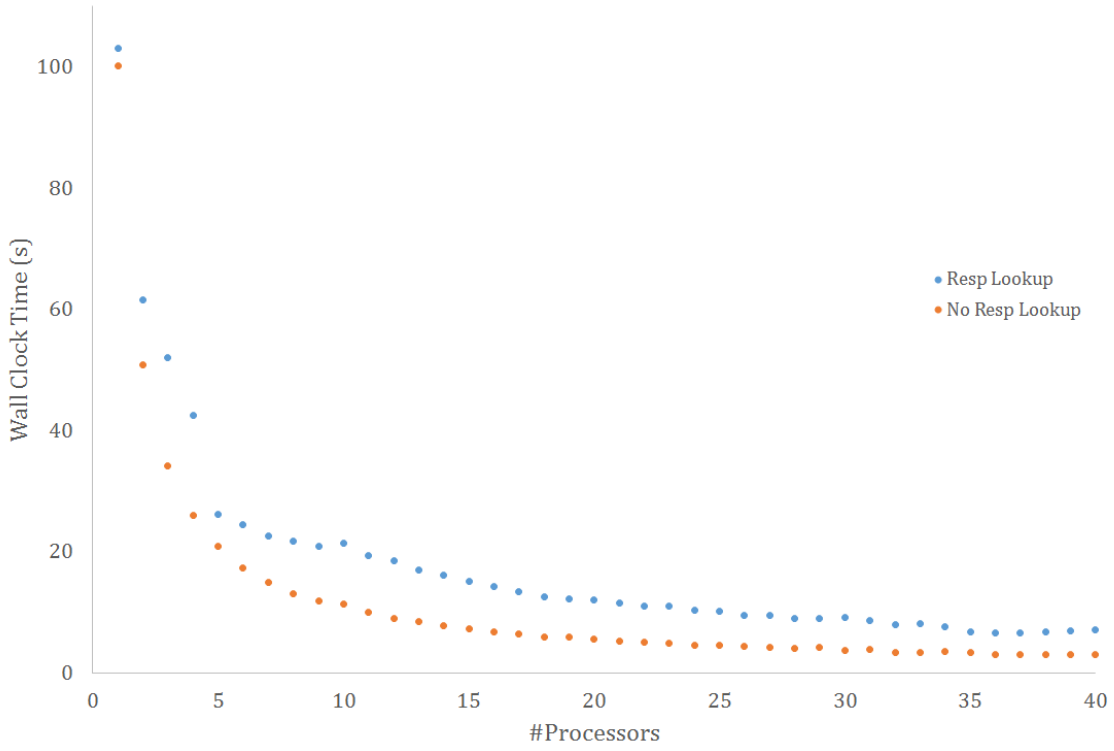

Figure 9. Wall-clock times for increasing number of processors for COMET-MPI solutions with and without response function lookup.

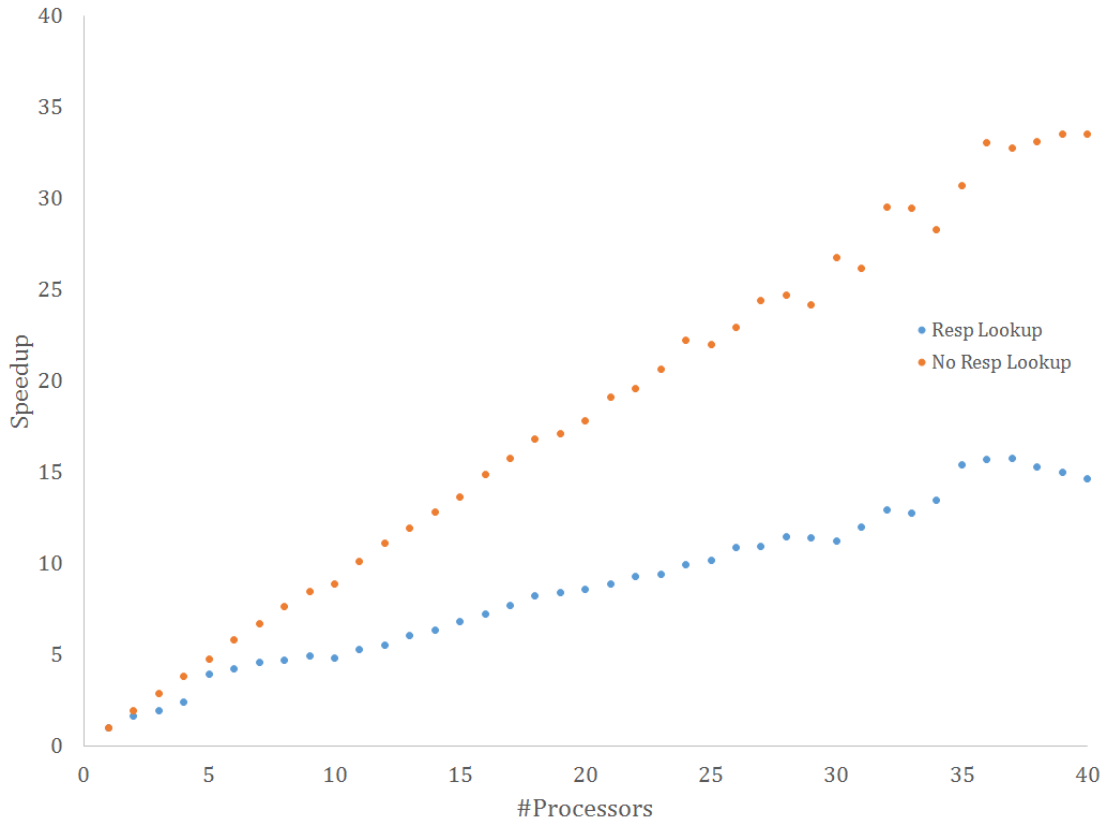

Figure 10. Speedups for increasing number of processors for COMET-MPI solutions with and without response function lookup.

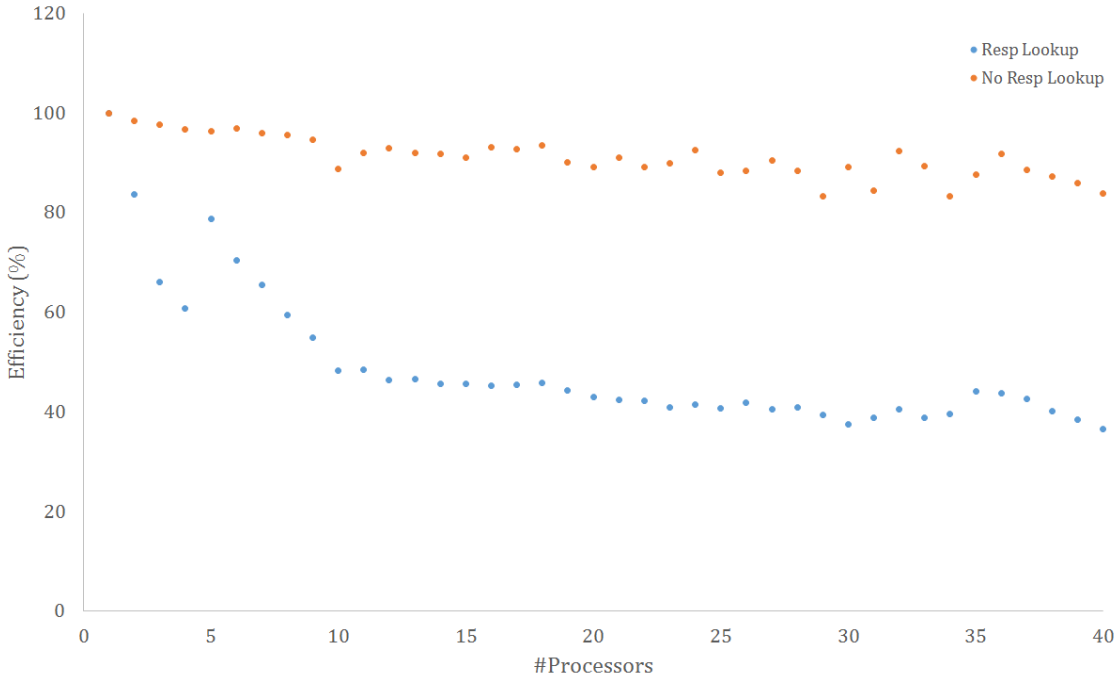

Figure 11. Parallel efficiencies for increasing number of processors for COMET-MPI solutions with and without response function lookup.

From the results seen in Figures 9-11, it is apparent that response function lookup can have a large impact on performance of the COMET-MPI code. For this problem, the case without response function lookups was about twice as efficient as the response function lookup case. Clearly, when analyzing performance of this code, consideration of lookup cost must be made in addition to analysis of calculation and communication costs.

5.2 Benchmarking Parallel Efficiencies with Theory

The impact of communication between processors on performance of parallel transport codes is a well-observed phenomenon [21]. However, the cost of frequent memory access and its effects on computational performance are often

less considered. To those experienced with parallel computing in neutron transport, the computational performance data presented in the previous subsection might even be considered counter-intuitive. Therefore, to ensure that the COMET-MPI code is behaving as expected, the results for both the response lookup case and the no response lookup case were benchmarked against predicted efficiencies given by Amdahl's Law.

As suggested by some literature [14,21,29], Amdahl's Law can be modified to reflect the cost of inter-processor communication:

$$S_p = \frac{1}{(1-f_p)+ {f_p}/{N}+ {T_C}/{T_S}}. \tag{43}$$

In the modified Amdahl's Law of equation (43), T_C is the time spent communicating between processors, and T_S is the serial execution time. Using this modified form (43) with estimated values for parallel fraction and communication cost, the approximate performance of a parallel code in terms of speedup and efficiency can be predicted. The actual performance of a code should agree with this prediction.

Kucukboyaci et al. [29] perform analysis with the modified Amdahl's Law to ensure that the neutron transport code PENTRAN behaves as expected. They vary the number of processors used in solving a simple problem and record the computational performance data. They compare their parallel efficiency results to those predicted by Amdahl's Law with an assumed parallel fraction and an estimated communication cost given by the median number of processors used in their paper. In this case, the *Amdahl efficiency* is given by

$$e_A = \frac{1}{\left(1-f_p+{}^{T_C}/_{T_S}\right)N+f_p}. \quad (44)$$

Equation (44) is obtained by dividing the speedup predicted by Amdahl's Law (43) by the number of processors used in a calculation. In equation (44), e_A is the Amdahl efficiency.

Analysis for the No Response Lookup Case

Analysis with Amdahl's Law was performed in a similar manner in this study. Since the case without response lookup demonstrates the effects of inter-processor communication on computational performance, the form of Amdahl's Law given by equation (43) was used to predict speedups and the form given by equation (44) was used to predict parallel efficiencies as the number of processors increased. Parallel fraction was estimated as

$$f_p = 1 - \frac{1}{CM*s*m}. \quad (45)$$

In equation (45), $CM * s * m$ is the size of one dimension of the total response matrix $R(k)$ in the inner iteration. This parameter is also the maximum decomposition that can occur on the problem using a combination of the decomposition techniques discussed in the previous chapter. Using the flux expansion, number of surfaces, and number of coarse meshes of the C5G7 problem given in the description in Chapter 6, this estimate resulted in a parallel fraction of approximately unity. The communication term, ${}^{T_C}/_{T_S}$, was estimated as 0.0062, which was the communication cost for the case with 20 processors. This was estimated by

$${}^{T_C}/_{T_S} = \frac{1}{N} - \frac{1}{S_p}. \quad (46)$$

Equation (46) shows the difference in inverse speedup between an Amdahl's Law with and without communication cost.

Using the assumed 20 processor communication cost in equation (44), the Amdahl efficiency was graphed against the observed parallel efficiencies for the case of no response lookups. The two are shown in Figure 12.

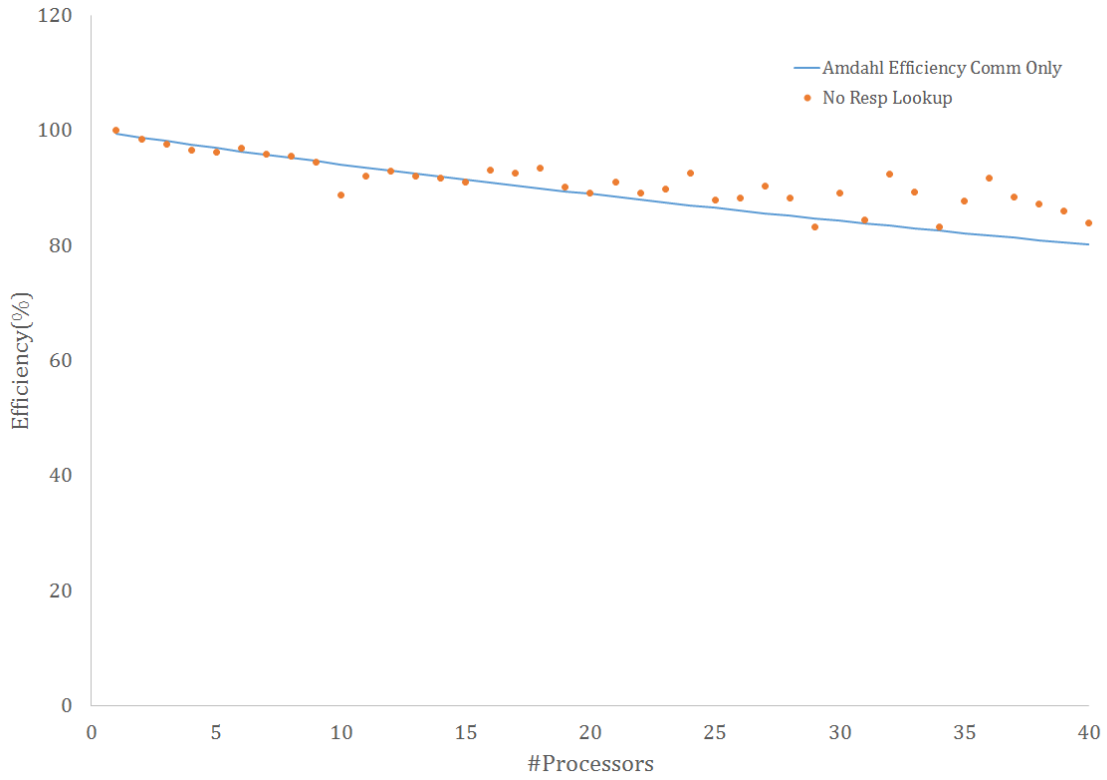

Figure 12. Amdahl efficiency and observed parallel efficiency for the no response lookup case.

As seen in Figure 12, the predicted Amdahl efficiencies agree well with the observed parallel efficiencies. The oscillation of efficiencies seen for the 21-40 processor run cases come from variations in the processor topologies of the decomposition, which are determined by the job scheduler on the cluster. Further, these runs involve processors on different compute nodes, which amplify this topology effect.

From these results, it can be concluded that when response lookup cost is neglected, the COMET-MPI code performs as expected when computation and communication costs dictate the performance of the code.

Analysis of the Response Lookup Case

Clearly, given the quite different behavior of the COMET-MPI code when response lookups are *not* neglected, different assumptions must be made when using Amdahl's Law to predict the performance of the code. One such approach is to include another term in Amdahl's Law to account for the cost of looking up response functions. Horoi and Enbody [30] follow this approach of adding a term to Amdahl's Law when considering effects outside of inter-processor communication. Another approach is to modify the assumed parallel fraction. From the perspective of predicting parallel efficiencies, these two approaches are equivalent, and the difference between the two is semantic. However, the difference does have an impact on analyzing the effects of response function lookup on improvement in computational performance from parallelization.

If lookup cost is considered as its own term in Amdahl's Law, such as in equation (47),

$$S_p = \frac{1}{(1-f_p) + {}^{f_p}/_{N} + {}^{T_C}/_{T_S} + {}^{T_{mem}}/_{T_S}}, \tag{47}$$

where ${}^{T_{mem}}/_{T_S}$ represents the cost of response function lookups, then this cost can be compared to the communication cost and its detriment to speedup. If lookup cost is considered within the parallel fraction, then the cost can be shown to directly impact parallelizability of a code. To demonstrate the difference between

these two approaches, both a response function lookup cost to be used in an Amdahl's Law analysis such as with equation (47) and a modified parallel fraction, which would be used Amdahl's Law analysis given by equations (43) and (44), are computed.

First, the response lookup cost itself was estimated. Using a similar approach to finding the communication cost at 20 processors, that is, finding the value of the lookup cost as the difference in inverse speedup between the response lookup and no response lookup cases, where the assumed speedups are of forms (47) and (43), respectively, the response lookup cost was computed as 0.0601. This cost is roughly 10 times larger than the estimated communication cost. This comparison shows directly that the cost of response function lookups is much greater and has a larger effect on performance of the COMET-MPI code than the inter-processor communication cost.

The modified parallel fraction was computed as well. The difference between inverse speedups of the response lookup and no response lookup cases with an Amdahl's Law of form (43) was computed. Using this difference in inverse speedup, an assumed parallel fraction of unity for the no response lookup case, and an assumed communication cost of 0.0062 on 20 processors, the modified parallel fraction for the case of response function lookup was 0.94. Clearly, this lower parallel fraction indicates that the parallelizability of the COMET-MPI code is lowered when response function lookups are involved in the analysis.

As mentioned previously, the difference between using equation (43) with a modified parallel fraction or using equation (47) with an assumed cost of

response function lookup to predict speedups (and therefore parallel efficiencies) is semantic – both approaches generate similar estimates for speedups and parallel efficiencies in that they produce similar predictions for speedup and parallel efficiency as a function of N, the number of processors used in a calculation Therefore, the approach that modifies the parallel fraction for equations (43) and (44) is used to predict Amdahl efficiencies for the response lookup case by preference. These predictions were compared against the observed parallel efficiencies for this case. The results are plotted in Figure 13.

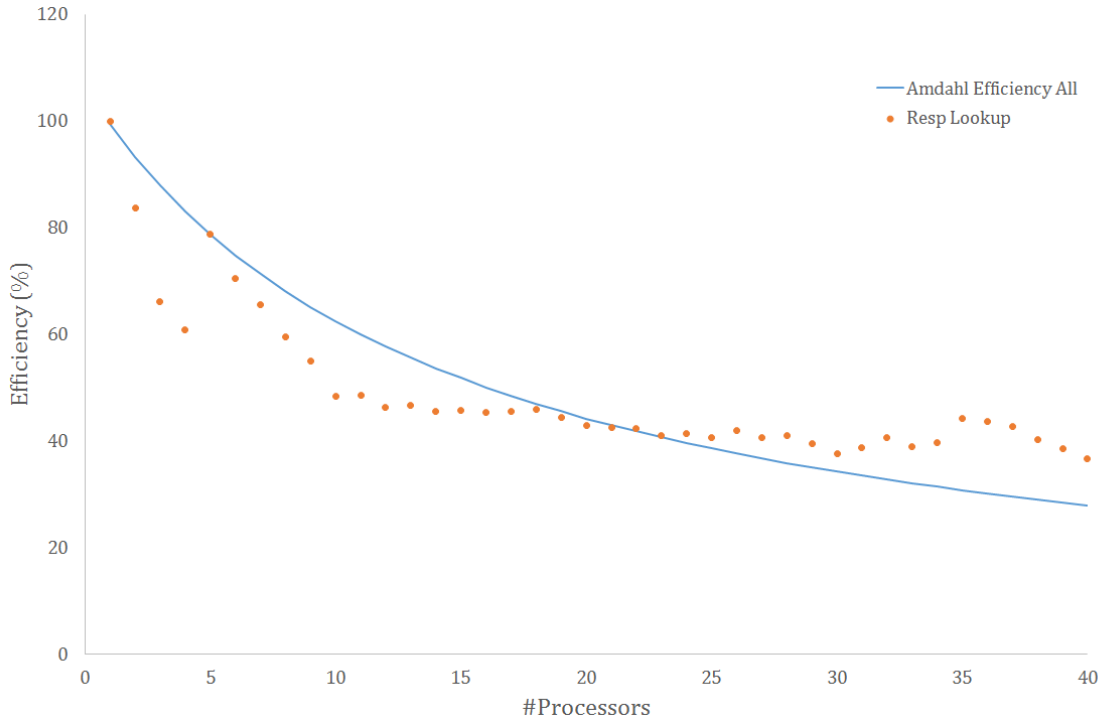

Figure 13. Amdahl efficiency and observed parallel efficiency for the response lookup case.

With the modified assumption in Amdahl's Law (43) or (47), the computational performance of the COMET-MPI code can be predicted from theory. The deviation from prediction for the case with low number of processors is due to decomposition

dependent response lookup costs degrading efficiency – response function lookup costs are really a function of the number of processors used in a calculation. After a sufficiently high number of processors is used in a calculation, the efficiencies agree much very well with theory. Therefore, despite the somewhat surprising effect of response function lookups impacting parallel performance as much as they do, the effect is to be expected when analyzing the performance of the code.

5.3 Discussion of Response Function Lookup Cost

Clearly, the cost of looking up response functions during the coarse mesh sweep of each inner iteration greatly affects the computational performance of the COMET-MPI code. From the results presented in this chapter, it can be concluded that the response function lookup cost, not communication cost, is the limiter on speedup and parallel efficiency of the code. The ramifications of this effect should be discussed in detail.

As an inner iteration progresses, it is a necessity to look up responses during the coarse mesh sweep. As outgoing expansion coefficients are updated via equation (35) in the coarse mesh sweep, different regions of a reactor are visited, which represent zones of varying state parameters. These state parameters will necessitate varying responses to accurately describe their behavior. As a result, when a region of a reactor is encountered with unique characteristics (e.g., material composition or state parameters such as temperature, material density, etc.), a new response function must be looked up to accurately compute the outgoing expansion coefficients for the coarse meshes. Despite their detriment to

computational performance, response function lookups are required for obtaining an accurate problem solution.

It is important to note that the effect of response function lookups is problem-dependent. Each problem has its own loading pattern of coarse meshes, so it should be concluded that each problem will have unique costs of response function lookups, even given a similar response function data set. The effects of different problems' coarse mesh loadings on computational performance is directly demonstrated in the next chapter, which shows COMET-MPI solutions for problems of varying size and coarse mesh loadings.

Further, different problems will have differing sizes of flux expansions. As the size of a flux expansion increases, the response function lookup cost will increase in kind. It should be noted that this works *against* the scalability of the COMET-MPI code as problems grow in flux expansion. This is in direct opposition to the prediction of the scalability model derived in the previous chapter, which indicated that problems with a high flux expansion will scale better with increasing number of processors. However, that model did not include effects of response function lookups, which have been shown to have a large impact on computational performance.

For the problems presented in this thesis, the cost of coarse mesh lookups might present a load *imbalance* in a calculation. While measures have been made in the algorithm to balance the load in relation to computational and communication effort, because of the geometric nature of the domain decomposition, some processors might have a greater number of unique coarse

meshes and therefore a higher cost of response function lookup than others. To remedy this load imbalance by redistributing coarse meshes among processors based upon response lookup cost is an interesting area of future work.

However, the problems presented in this thesis are mostly snapshot-in-time calculations for idealized benchmark problems. In the future, it is desired to compute COMET solutions coupled to burnup calculations. For these types of problems, the number of unique coarse meshes in a sweep approaches the total number of coarse meshes in a problem. This is because, while there might exist only a few types of assemblies in a core design (e.g., 3.5 w% UO_2, 2 w% UO_2 with 6 w% Gd_2O_3), each region of the core will burn differently. Therefore, the material compositions of each coarse mesh will vary. For this class of problems, the load imbalance of disparity in response function lookups between processors is significantly diminished, since every coarse mesh in a sweep will require a new lookup.

Because of the effect of the response function lookup cost on computational performance of the COMET-MPI code, future work on developing parallel implementations of COMET should be focused on ways of improving the response function lookups' use of memory bandwidth to improve computational performance and parallel efficiency.

CHAPTER 6

SENSITIVITY STUDY OF PROBLEM SIZE ON SCALABILITY OF THE COMET-MPI CODE

The effects of problem size on parallel efficiency were studied in the scope of this thesis. Using variants of the C5G7 problem, number of coarse meshes and size of flux expansions were varied, and the parallel performance of COMET-MPI was recorded. This was to assess the varying effects of computational costs, communication costs, and, very importantly, response function lookup costs as the problem being studied changed in size. In all cases, COMET-MPI runs were carried out on a dedicated homogeneous cluster of Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz processors, with 2 nodes of 20 processors each.

6.1 Description of the Benchmark Problems

In the sensitivity studies, problems used were variants of the C5G7 problem [28], which has both UO_2 and MOX fuel. The first variant is the C5G7 problem with vacuum boundary conditions, henceforth called the "1X C5G7 problem." A radial view of the problem is provided in Figure 14.

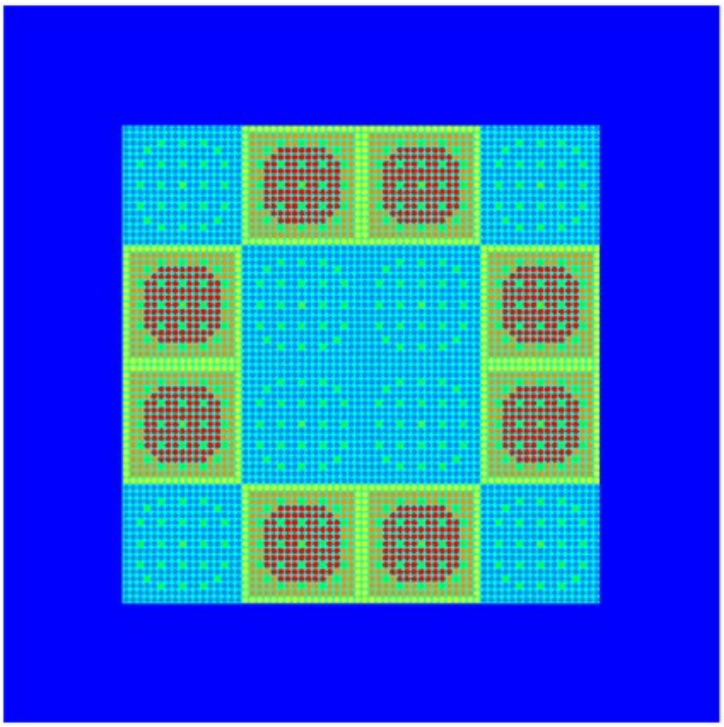

Figure 14. Radial view of the 1X C5G7 Problem.

The second problem used in the sensitivity study is the "2X C5G7 Benchmark Problem," so called because it has two repetitions of the C5G7 active core on a side. A radial view of this problem is provided in Figure 15.

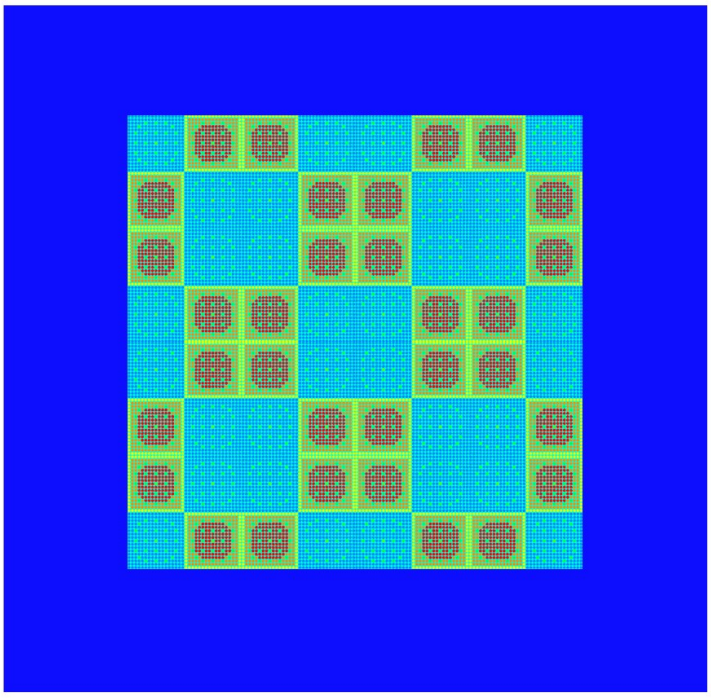

Figure 15. Radial view of the 2X C5G7 Problem.

The third and final problem used in the sensitivity study is the "3X C5G7 Problem," due to its repetition of three C5G7 active cores along a side. The radial view of this problem is given in Figure 16.

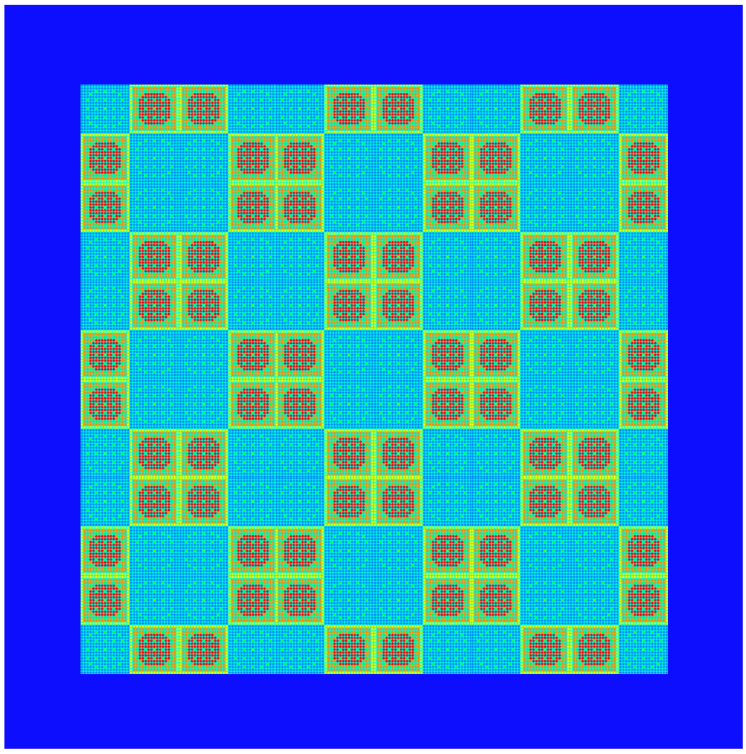

Figure 16. Radial view of the 3X C5G7 Problem.

COMET Models

Computational models of these problems for use in the COMET code were built and used when monitoring computational performance. The models used here are similar to those used in previous studies [7]. Using the seven energy group cross section library provided in the reference, a response library was built with expansions of fourth order in space and second order in angle using the stochastic code MCNP5 [24]. 50 million particles were run per case, and responses measuring surface angular flux, pin fission density, neutron production, and neutron absorption were tallied. Seven unique coarse meshes were modeled, and they are given in Table 1.

Table 1. The unique coarse meshes modeled for the benchmark problems.

	Mesh	Dimension
1	UO_2 Uncontrolled Assembly	21.42” x 21.42” x 14.28”
2	UO_2 Controlled Assembly	21.42” x 21.42” x 14.28”
3	MOX Uncontrolled Assembly	21.42” x 21.42” x 14.28”
4	MOX Controlled Assembly	21.42” x 21.42” x 14.28”
5	Upper Unrodded Reflector	21.42” x 21.42” x 21.42”
6	Lower Unrodded Reflector	21.42” x 21.42” x 14.28”
7	Upper Rodded Reflector	21.42” x 21.42” x 21.42”

In Table 1, the reflector coarse meshes are filled completely with water and are homogeneous with the exception of the “Upper Rodded Reflector,” which has control rods inserted. The pin cell makeup of the fuel coarse meshes is given in Figures 17 and 18. While the models have explicit fuel-moderator heterogeneity, this is not reflected in the figures.

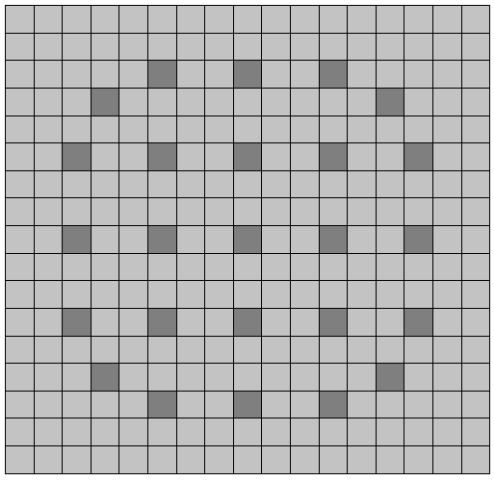

Figure 17. Pin cell makeup of the UO_2 assembly coarse meshes, from [8].

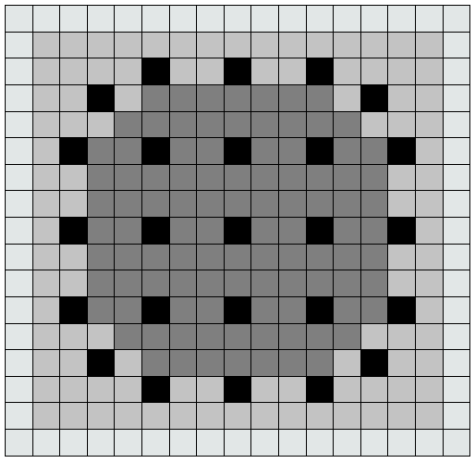

Figure 18. Pin cell makeup of the MOX assembly coarse meshes, from [8].

In Figure 17, the lightly shaded areas are pin cells with UO_2 fuel rods. The darkly shaded areas are guide tubes in the uncontrolled mesh and are control rods in the controlled mesh. The exception is the center pin cell, which is always a fission chamber. In Figure 18, the lightly shaded areas are pin cells with 4.3% MOX, the slightly darker areas pin cells with 7.0% MOX, and the dark grey areas are pin cells with 8.7% MOX. The black areas are guide tubes in the uncontrolled mesh and are control rods in the controlled mesh; as with the UO_2 mesh, the center pin cell is always a fission chamber.

All problems use these unique coarse meshes and their precomputed data as their computational models. In addition, vacuum boundary conditions are assumed in all cases. The difference between the problems is the coarse mesh loading patterns. Radially, the Figures 14-16 demonstrate how the coarse mesh loadings vary between the different problems. The problems also have different axial loadings, which are shown in Table 2, which also shows how many total coarse meshes are used for each case.

Table 2. Axial and total coarse mesh loadings of the three benchmark problems.

	Active/Total Axial CM Loading	Total CM Loading
1X	6/8	288
2X	12/14	2016
3X	16/18	4608

In Table 2, the distinction between active and total axial coarse mesh loading is made because the top and bottom axial zones of each problem are loaded with coarse mesh type 7 (from Table 1) where fuel is placed in the active core (every axial zone that is not the top or bottom zone). In the top and bottom axial zones, the radial reflector meshes are modeled by coarse mesh type 5 and are modeled by coarse mesh type 6 in all other axial zones. For the 1X C5G7 Problem, one radial reflector coarse mesh is used on the perimeter of every axial zone. For the 2X and 3X C5G7 Problems, two reflector coarse meshes are used on the perimeter of every axial zone.

6.2 Number of Coarse Meshes Sensitivity Study

To assess the effects of changing the number of coarse meshes in a problem to the scalability of the code, the 1X, 2X, and 3X C5G7 problems were solved with COMET-MPI on various numbers of processors, varying from 1 to 40. When solving these problems, the number of iterations performed was fixed – one outer iteration to evaluate the response functions and 150 inner iterations, mirroring a typical set of inner iterations in a core solve. The number of iterations is held fixed to assess the change in speedup for various numbers of processors for problems of

various sizes. Chebyshev polynomial filtering was used in the calculations. Computational performance was only measured for the inner iterations, to evaluate the scalability model from Chapter 4 and to assess the effects of response function lookup as the problem size increased. The results of each calculation are given in Tables 3-5. When examining computational performance data for parallel programs, it is often instructive to view the speedup and parallel efficiency data graphically. The speedups and parallel efficiencies are plotted in Figures 19 and 20, respectively.

Table 3. Parallel performance results for the 1X C5G7 Problem; COMET calculated core eigenvalue $k = 1.14367 +/- 2\ pcm$

#Processors	Wall Clock Time (s)	Speedup	Efficiency(%)
1	101	1.0	100
5	26	3.9	78
10	20	5.1	51
20	11	9.2	46
40	6	16.8	42

Table 4. Parallel performance results for the 2X C5G7 Problem; COMET calculated core eigenvalue $k = 1.22163 +/- 1\ pcm$

#Processors	Wall Clock Time (s)	Speedup	Efficiency(%)
1	701	1.0	100
5	155	4.5	90
10	102	6.9	69
20	52	13.5	68
40	28	25	63

Table 5. Parallel performance results for the 3X C5G7 Problem; COMET calculated core eigenvalue $k = 1.23971 +/- 1\ pcm$

#Processors	Wall Clock Time (s)	Speedup	Efficiency(%)
1	1639	1.0	100
5	356	4.6	92
10	215	7.6	76
20	111	14.8	74
40	56	29.3	73

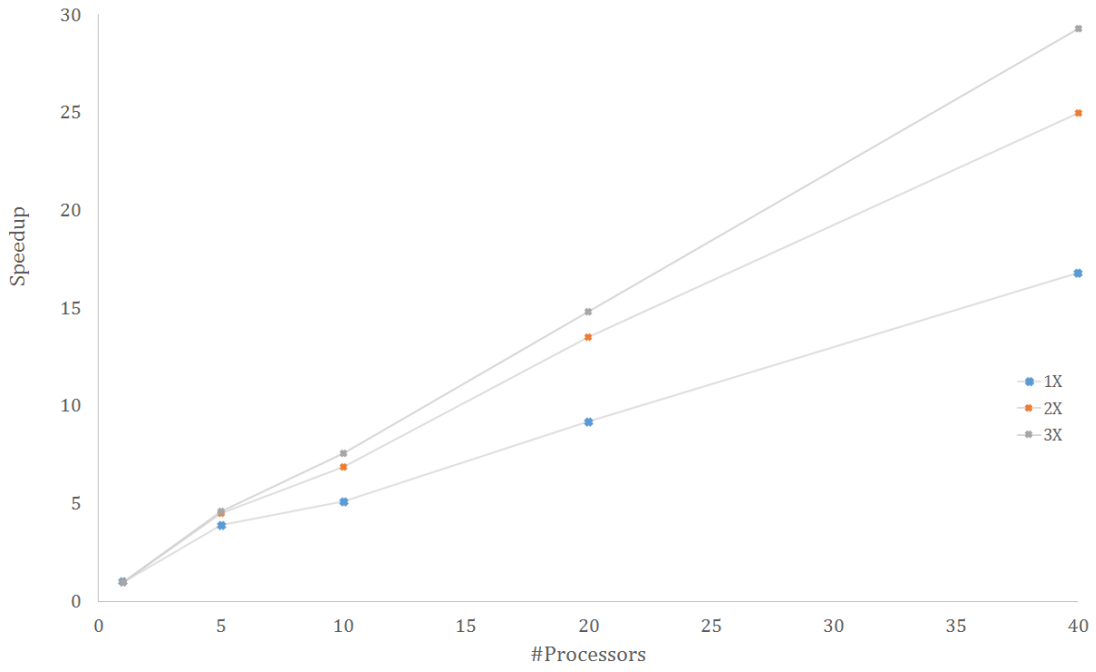

Figure 19. Speedups for the 1X, 2X, and 3X C5G7 problems for increasing numbers of processors.

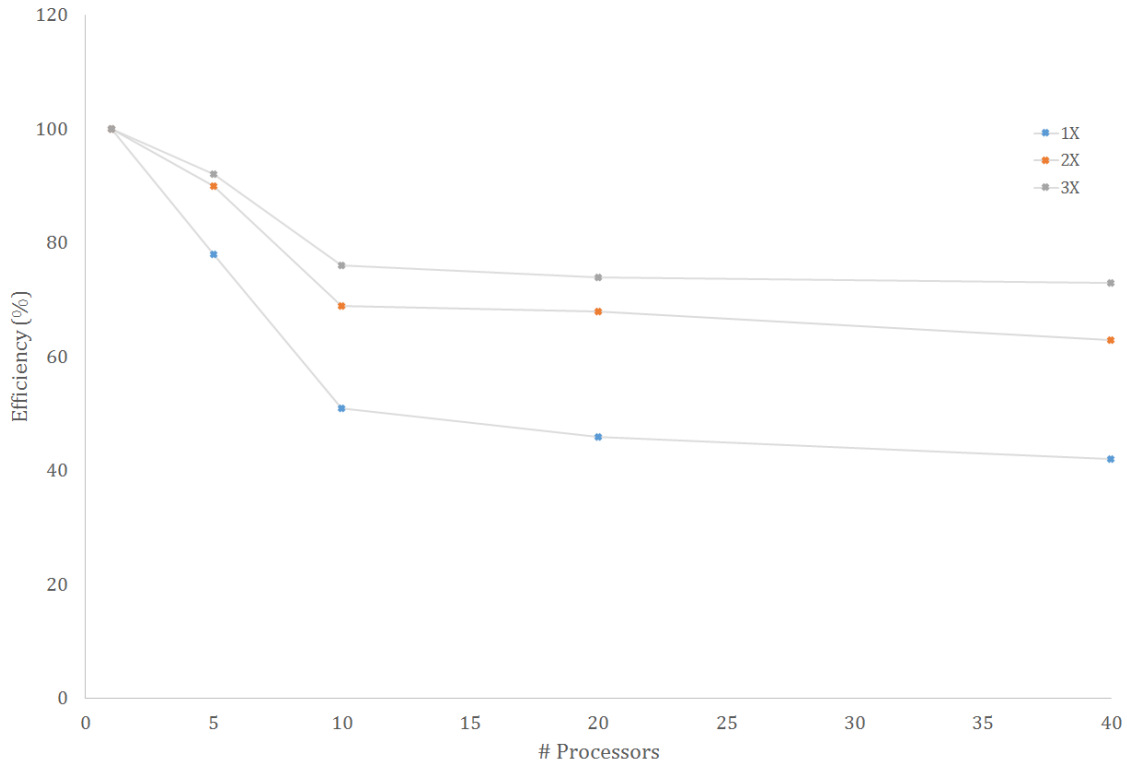

Figure 20. Parallel efficiencies for the 1X, 2X, and 3X C5G7 problems for increasing numbers of processors.

As indicated by Tables 3-5 and demonstrated graphically in Figure 20, parallel efficiency increases with increasing number of coarse meshes in a calculation. This agrees with the scalability model derived in Chapter 4. The results also demonstrate that the degradation in parallel efficiency due to response function lookup varies between problems and can decrease for increasing problem size given the same number of processors used to calculate a solution.

These encouraging results imply that for problems with large numbers of coarse meshes, high parallel speedups and efficiencies can be obtained. An application of this could be finer axial meshing of core calculations with the COMET method. It should be noted that when a calculation increases from

utilizing 5 to 10 processors, efficiency for all problems decreases, while speedups scale nearly linearly with other changes in processors. This effect was demonstrated for the C5G7 problem in the previous chapter and is a consequence of reduced parallel efficiency due to response function lookup costs.

6.3 Flux Expansion Sensitivity Study

To assess the effects of changing the flux expansion in a problem to the scalability of the code, the 3X C5G7 problem was solved with COMET-MPI using various flux expansions ranging from zeroth order in both space and angle to the maximum flux expansion. For each flux expansion, the problem was solved on 1 and 40 processors. As with the size sensitivity study, when solving these problems, the number of iterations performed was fixed – one outer iteration to evaluate the response functions and 150 inner iterations. Again, Chebyshev polynomial filtering was used in the calculations. Computational performance was only measured for the inner iterations to evaluate the scalability model. The results of each calculation are given in Table 6. As with Tables 3-5, the wall clock times in Table 6 are given in seconds, and efficiency is reported in percent.

Table 6. Parallel performance results for the 3X C5G7 Problem for various flux expansion orders.

Flux Expansion	Wall Clock Time - 1	Wall Clock Time - 40	Speedup	Efficiency
0,0,0,0	16 s	0.7 s	23	58 %
2,2,0,0	40 s	1.4 s	29	73 %
2,2,2,2	631 s	18 s	35	88 %
4,4,2,2	1639 s	56 s	29	73 %

The results of this sensitivity study are of particular interest. Increasing the flux expansion order from zeroth order up to second order in space and angle increases the parallel efficiency. This agrees with the scalability model derived in Chapter 4. However, increasing the flux expansion order to the maximum in the database reveals a degradation in computational efficiency. This is an effect of the response function lookup cost on the parallel efficiency of the COMET-MPI code. Further, from this sensitivity study, it is demonstrated that large flux expansions (and thus large response submatrices) increase the magnitude of this effect. How this applies to larger problems (full core problems utilizing a high flux expansion) is the focus of a future chapter.

While there are some effects that the scalability model from Chapter 4 did not account for, it is seen that increasing the number of coarse meshes does increase the scalability of COMET-MPI. In addition, the problem-dependent nature of the response function lookup costs is demonstrated When increasing flux expansion, an increase in parallel efficiency is also observed to a point. The increase in parallel efficiency does saturate and start to yield negative returns due

to response function lookup costs. With some knowledge of how COMET-MPI behaves for various problem cases, it is of interest to compute whole-core solutions with the code to observe how it performs. This is the focus of the next chapter.

CHAPTER 7

WHOLE CORE BENCHMARK PROBLEM SOLUTIONS WITH COMET-MPI

The sensitivity study results of the previous chapter show that even on a relatively small research cluster, computational gains in COMET calculations by using the COMET-MPI code can extend up to a factor of 35 speedup over a serial COMET calculation. With these encouraging preliminary results as motivation, the focus of study shifted to solving whole core benchmark problems with COMET-MPI. Two benchmark problems were solved, and one of the benchmark problems were solved with different sets of nuclear data. The benchmark problems, computational models, and results are described in this chapter.

7.1 Description of the Benchmark Problems

Two benchmark problems were solved in this section of the study: a PWR with gadolinium benchmark problem [31] and a PWR with MOX fuel benchmark problem [32]. These problems were selected because the COMET method has been verified against these problems [3], so it is known that the COMET solutions for these problems agree well with Monte Carlo benchmarks. In addition, these PWR problems offer sufficient meshing and heterogeneity to reflect realistic problems that the COMET method has encountered and will continue to solve in the future.

PWR with Gadolinium

This benchmark problem is comprised of 193 square fuel assemblies in the active core, with length 21.505 cm on a side. Standard fuel assemblies in the reactor core contain UO_2 fuel pins, enriched to 4.1% by weight of U-235. There are also assemblies which contain fuel rods doped with gadolinium (Gd_2O_3) as a burnable absorber. In such fuel rods, the fuel is enriched to 2.6% by weight of U-235, and the fuel pins contains 6% gadolinium by weight. The benchmark core also features assemblies at different burn levels: fresh fuel, 15 GWd/T (once burned), 33 GWd/T (two times burned), and 50 GWd/T (three times burned). The assumed boundary condition for this problem is a vacuum boundary.

A radial cross section of the core is illustrated in Figure 21, where the different colors designate different types of fuel assemblies. Figure 22 shows the axial meshing of the core used in the modelling of the problem, which includes 17 zones along the active core length. Figures 23 and 24 demonstrate the pin cell makeup of the fuel assemblies with and without gadolinium. Even though not explicitly shown in Figure 23-24, note that full fuel-clad-moderator heterogeneity is modelled on the pin cell level. In these figures, the white regions are UO_2 pin cells, the grey regions are gadded UO_2 pin cells, the black regions are cell with either guide tube or inserted control rods.

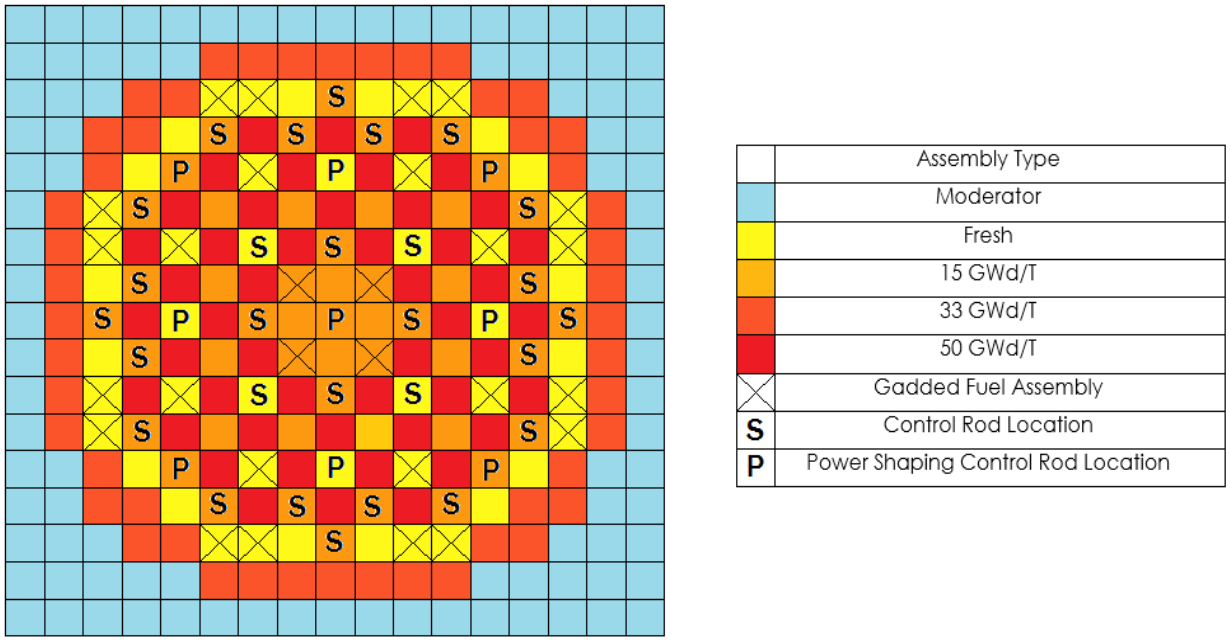

Figure 21. Radial core layout of the PWR with gadolinium benchmark problem, from [9].

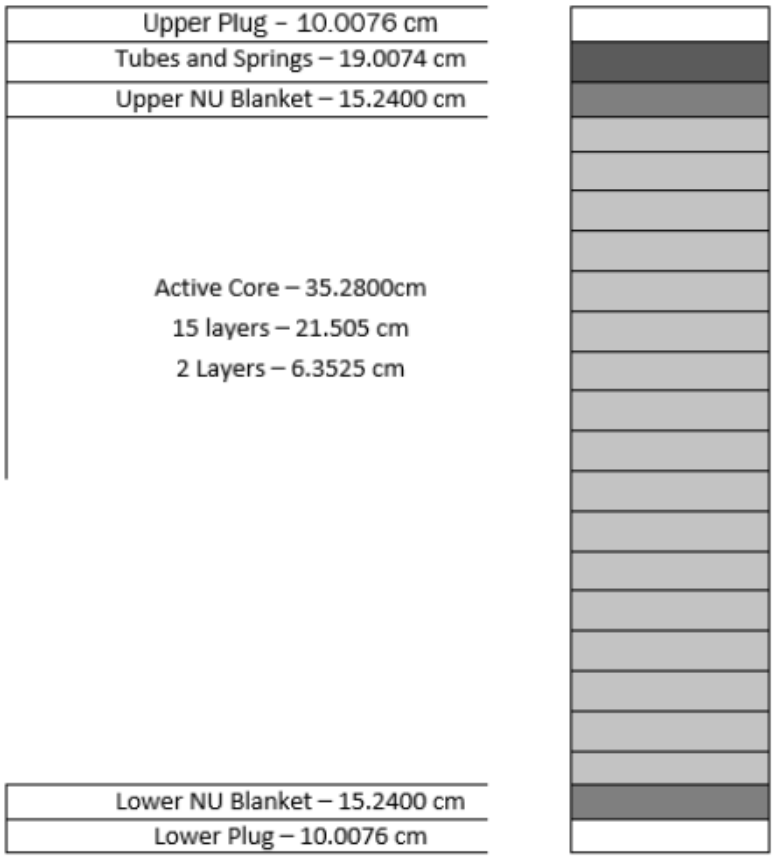

Figure 22. Axial meshing of the PWR w/gad core used in the benchmark problem, from [9].

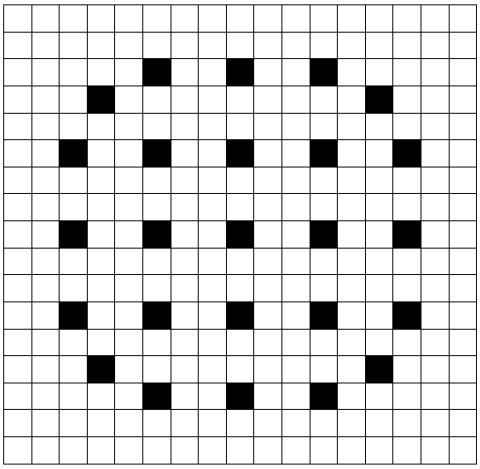

Figure 23. Pin cell layout of a UO_2 mesh without gadolinium.

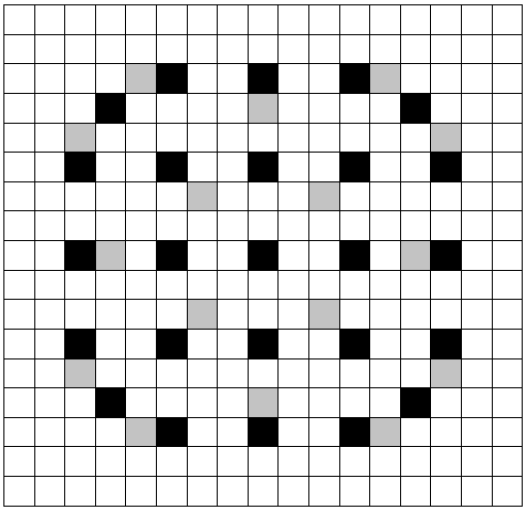

Figure 24. Pin cell layout of a UO_2 mesh with gadolinium.

The problem contains three core configurations which depend upon insertion of control rods: all-rods-in (ARI), all-rods-out (ARO), and some-rods-in (SRI). Control rod insertion into assemblies is indicated by Figure 21 by the letters "S" and "P." In the SRI configuration, only control rods in the assemblies marked "P" will have control rods inserted. In the ARI configuration, all control rods are inserted into the core (where there are assemblies marked with letters). In the ARO configuration, no control rods are inserted. For the case of this study, only the ARO solution is computed.

The computational model for this problem utilized a two energy group library provided in the reference [31]. Again using the stochastic code MCNP5, a response library was computed using 50 million particles per case. A maximum flux expansion of sixth order in space and second order in angle was computed. Responses tallied were surface flux, pin fission density, neutron production, and

neutron absorption. The computational model utilizes 39 unique coarse meshes, which are briefly described in Table 7.

Table 7. Unique coarse mesh specification for PWR w/Gad computational model.

Mesh	Coarse Mesh Type	Mesh	Coarse Mesh Type
1	UO_2 uncontrolled, fresh, full size	21	UO_2 gadded plug, fresh
2	UO_2 controlled, fresh, full size	22	UO_2 uncontrolled plug, once burned
3	UO_2 gadded, fresh, full size	23	UO_2 controlled plug, once burned
4	UO_2 uncontrolled, once burned, full size	24	UO_2 gadded plug, once burned
5	UO_2 controlled, once burned, full size	25	UO_2 uncontrolled plug, twice burned
6	UO_2 gadded, once burned, full size	26	UO_2 uncontrolled plug, thrice burned
7	UO_2 uncontrolled, twice burned, full size	27	UO_2 uncontrolled tube spring, fresh
8	UO_2 uncontrolled, thrice burned, full size	28	UO_2 controlled tube spring, fresh
9	UO_2 uncontrolled, fresh, small size	29	UO_2 gadded tube spring, fresh
10	UO_2 controlled, fresh, small size	30	UO_2 uncontrolled tube spring, once burned
11	UO_2 gadded, fresh, small size	31	UO_2 controlled tube spring, once burned
12	UO_2 uncontrolled, once burned, small size	32	UO_2 gadded tube spring, once burned
13	UO_2 controlled, once burned, small size	33	UO_2 uncontrolled tube spring, twice burned
14	UO_2 gadded, once burned, small size	34	UO_2 uncontrolled tube spring, thrice burned
15	UO_2 uncontrolled, twice burned, small size	35	Moderator 1
16	UO_2 uncontrolled, thrice burned, small size	36	Moderator 2
17	UO_2 uncontrolled blanket, fresh	37	Moderator 3
18	UO_2 controlled blanket, fresh	38	Moderator 4
19	UO_2 uncontrolled plug, fresh	39	Moderator 5
20	UO_2 controlled plug, fresh		

PWR with MOX

This benchmark is a PWR full-core benchmark problem with both UO_2 and MOX fuel from Rahnema et al [32]. The benchmark specification is the same in this study as given in the reference, except that here the core was divided into sixteen axial zones along the active core length, as in Remley and Rahnema [8] rather than four. The active core is arranged in a checkerboard pattern of 121 fuel assemblies surrounded by a water reflector. Vacuum boundary conditions are assumed. A radial cross section of the core and the axial layout are given in Figures 25 and 26, respectively.

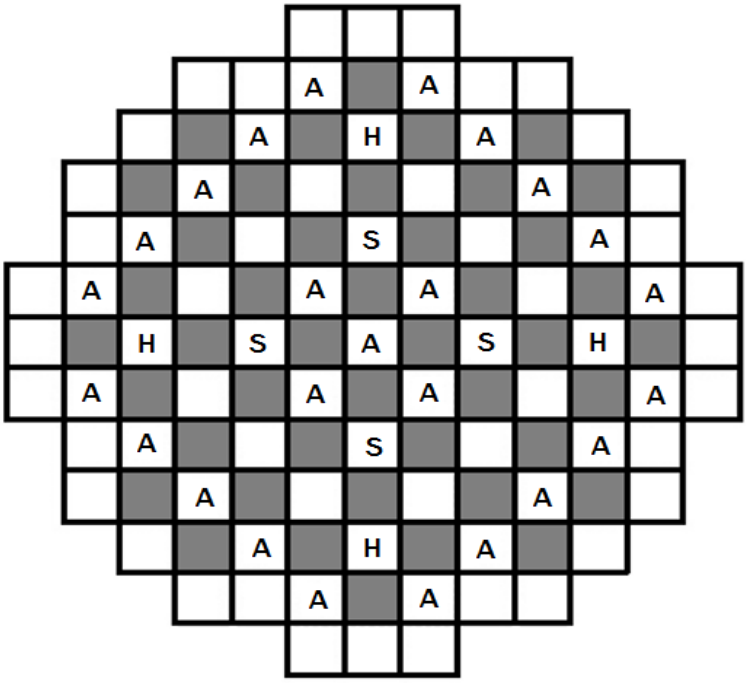

Figure 25. Radial active core layout of the PWR with MOX, from [8].

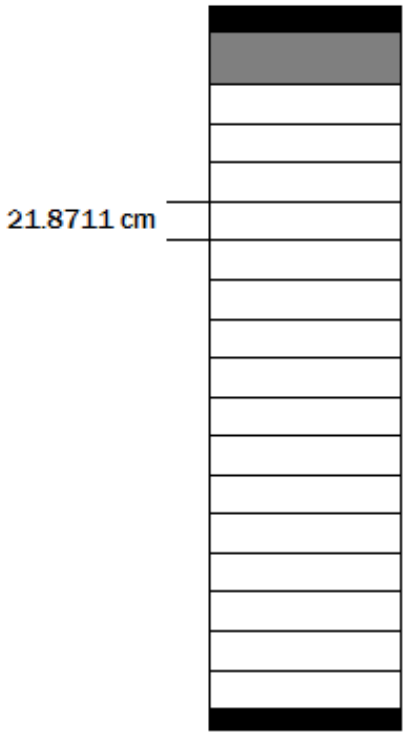

Figure 26. Axial meshing used in the PWR with MOX benchmark, from [8].

In figure 25, the unshaded regions represent UO_2 fuel assemblies, and shaded regions represent MOX fuel assemblies. The regions marked with "A," "S," and "H" are assemblies where control rods may or may not be inserted, depending upon core configuration. In Figure 26, the unshaded regions represent are a division along the active core length. As shown in Figure 16, all lengths of the axial zones in the active core are 21.8711 cm. The black-shaded regions represent the plug, and the grey-shaded area represents the reactor's tube-spring. The pin cell layout of these assemblies is the same as the pin cell layout in the 1-3X C5G7 problems solved the in previous chapter. However, in this problem, the heterogeneity of the fuel, clad, and moderator are explicitly modeled. Unlike in the

previous full core benchmark problem, all the fuel in this problem is assumed to be fresh (never burned) fuel. Table 8 shows some geometric parameters in the benchmark specification.

Table 8. Geometric Specifications of the PWR with MOX benchmark.

Parameter	
Number of fuel pins per assembly	264
Number of control rods/guide tubes	24
Fuel radius	0.4095 cm
Cladding outer radius	0.54 cm
Guide tube inner radius	0.573 cm
Guide tube outer radius	0.63 cm
Pin pitch	1.26 cm
Assembly pitch	21.42 cm

In the benchmark specification, there are three possible fuel assembly types, based upon control rod insertion: UO_2 controlled, UO_2 uncontrolled, and MOX uncontrolled. As with the PWR with gadolinium, different configuration of this core depend on control rod insertion. The three core configurations in the benchmark specification are All-Rods-Out (ARO), All-Rods-In (ARI), and Some-Rods-In (SRI). In ARO configuration, all rods are fully removed from the core. SRI indicates control rods in assemblies marked “S” in figure 25 to be fully inserted, and control rods in assemblies marked “H” in figure 25 to be halfway inserted. ARI configuration indicates full control rod insertion in all assemblies marked either

"A," "S," or "H" in figure 18. As with the PWR with gadolinium problem, only the ARO configuration is solved in this study.

The computational model for this problem utilizes two different energy group data sets that are specified in the reference [32]. A response library was compiled using a two energy group data set, and a different response library was compiled using an eight energy group data set. Both libraries were compiled to study the effect of increasing energy groups on the efficiency of the calculations. In the two group case, a maximum flux expansion of fourth order in space and fourth order in angle was used. In the eight group case, a maximum flux expansion of fourth order in space and second order in angle was used. In both cases, 50 million particles were run per response case, and surface flux, pin fission density, neutron production, and neutron absorption responses were tallied. Nine unique coarse meshes were used to model the problem (for both energy group cases). The unique coarse mesh descriptions are given in Table 9.

Table 9. Unique coarse mesh specifications for the PWR with MOX benchmark.

Mesh	Description
1	UO_2 Controlled Assembly
2	UO_2 Uncontrolled Assembly
3	MOX Uncontrolled Assembly
4	Tube/Spring Uncontrolled Assembly
5	Tube/Spring Controlled Assembly
6	Plug Assembly
7	Active Core Reflector
8	Tube/Spring Reflector
9	Plug Reflector

7.2 Benchmark Solutions

Solutions to the whole-core benchmarks described in this chapter were calculated using COMET-MPI. In contrast to the computations in the previous chapter, where iterations were held fixed, the solutions in this section are fully converged results. In all calculation cases, both LOA and Chebyshev polynomial filtering are used. As with the calculations carried out in the previous chapter, the COMET-MPI runs in this section were carried out on a dedicated homogeneous research cluster of Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz processors, with 2 nodes of 20 processors each.

PWR with Gadolinium Solution

Solutions to the PWR with Gadolinium problem were computed for the full sixth order in space and second order in angle flux expansion. The COMET-MPI-calculated core eigenvalue for this problem was $k = 1.05323 +/- 1\,pcm$. COMET-MPI computed the solution using various numbers of processors, and the runtime data was recorded. Table 10 shows the wall clock times for calculation of the solution for increasing numbers of processors, given for convenience in minutes.

Table 10. Runtime results for the PWR with gadolinium benchmark.

#Processors	Wall Clock Time (min)	Speedup	Efficiency(%)
1	39	1.0	100
5	10	3.9	78
10	6.2	6.2	62
20	3.6	11	55
40	2.1	19	48

The results in Table 10 are encouraging. When solving the benchmark problem on 40 processors, a whole-core solution is computed in just over *two minutes*. This high level of computational efficiency is obtained, it should be noted, using a larger flux expansion than previously used in COMET calculations for this problem [3]. Table 10 gives the total execution time of COMET-MPI for these whole core solves. However, it is instructive to also look at the wall clock times for the inner iterations. The runtime data for the LOA inner iterations is given in Table 11, and the runtime data for the full order inner iterations is given in Table 12. Since speedup and efficiency are the quantities of interest for these data, the wall clock times are reported in seconds. Further, the speedups and parallel efficiencies of the whole solve, the LOA inner iteration, and the full order inner iteration are demonstrated graphically in Figures 27 and 28, respectively.

Table 11. Computational performance results for LOA inner iteration for PWR with gad

#Processors	Wall Clock Time (s)	Speedup	Efficiency(%)
1	109	1.0	100
5	24	4.5	90
10	14	7.8	78
20	9	12	60
40	6	18	45

Table 12. Computational performance results for full-order inner iteration for PWR with gad

#Processors	Runtime(s)	Speedup	Efficiency(%)
1	699	1.0	100
5	172	4.1	82
10	111	6.3	63
20	59	12	60
40	33	21	55

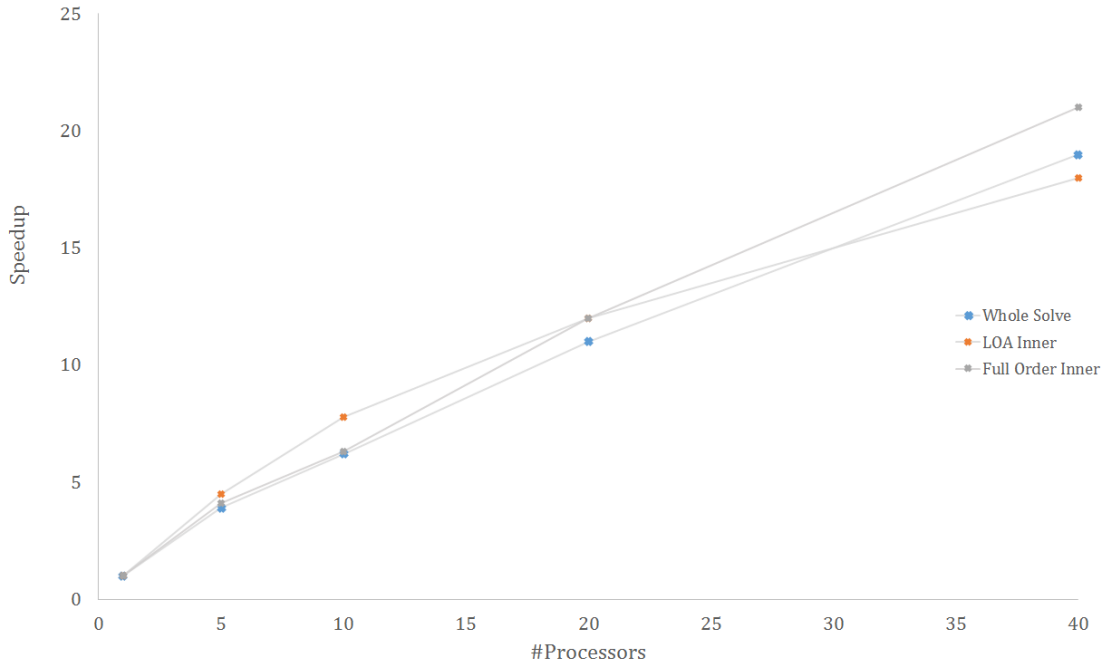

Figure 27. Speedups for the whole solve, LOA inner iteration, and full order inner iteration for the PWR with gadolinium problem.

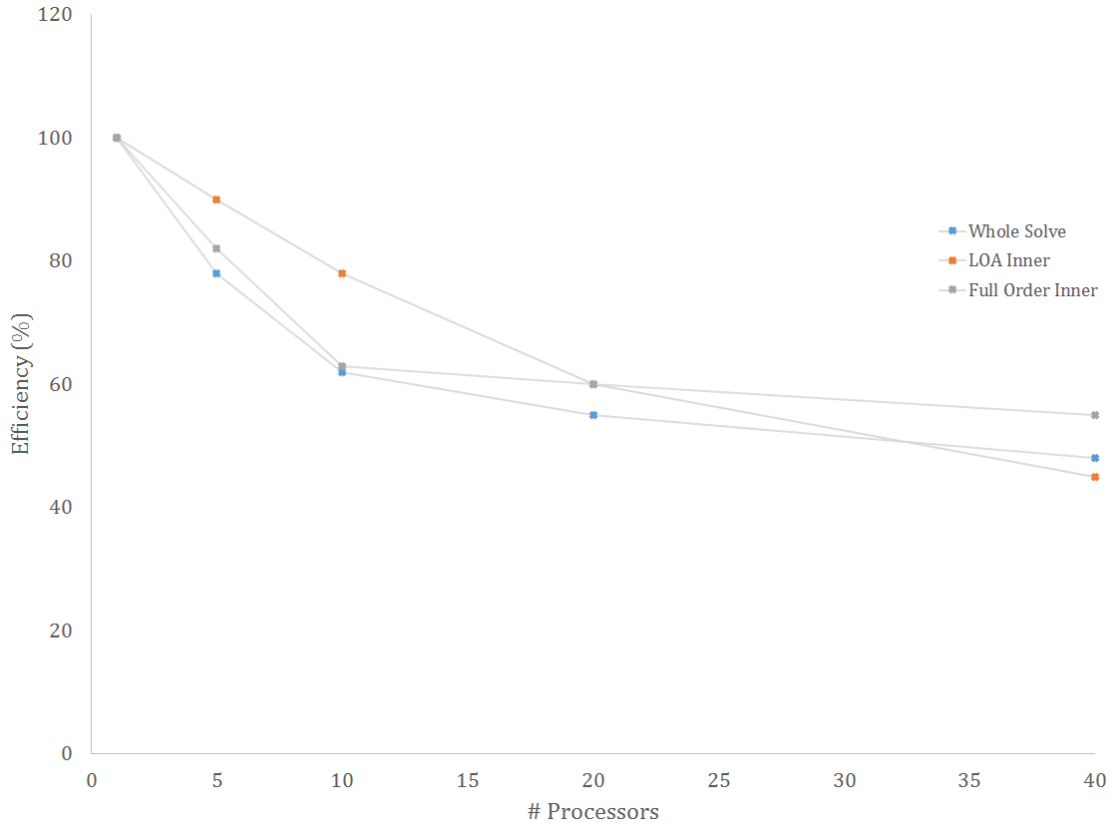

Figure 28. Parallel efficiencies for the whole solve, LOA inner iteration, and full order inner iteration for the PWR with gadolinium problem.

Tables 11-12 and Figures 27-28 indicate that the computational performance behavior for the whole core solve is driven by the computational performance of the inner iteration convergence. This is an expected result, as it has been stated that the inner iteration is the bulk of the computational effort in COMET calculations. Another observation to make from the results for this case is the higher order result has more parallel efficiency up to 40 processors than the LOA result, which agrees with the scalability model of chapter 4. This indicates that response function lookup cost for this problem is similar to the communication cost for this problem, in contrast to the problems of the previous

chapters. Further, Figure 28 shows parallel efficiency behavior that agrees with the predictions of Amdahl's Law as demonstrated in Chapter 5.

PWR with MOX – Two Energy Groups

Solutions for the PWR problem with MOX in two energy groups were computed with COMET-MPI utilizing the full fourth order in space and angle expansion. The calculated eigenvalue for these calculations was $k = 1.02655 +/- 1\ pcm$. The computational performance of the solutions for various numbers of processors in a calculation was recorded. Tables 13-15 show the computational performance for full core solve, the LOA inner iteration, and the full order inner iteration, respectively. As with the previous case, the wall clock times in Table 13 are most important and are given in minutes while the wall clock times in Tables 14-15 are reported in seconds. Figures 29 and 30 give plots of the speedups and parallel efficiencies of the whole solve, the LOA inner iteration, and the full order inner iteration.

Table 13. Computational performance results for the full core PWR w/MOX 2g solution

#Processors	Wall Clock Time (min)	Speedup	Efficiency(%)
1	52	1.0	100
5	12	4.5	90
10	6.7	7.8	78
20	3.5	15	75
40	2.0	25	63

Table 14. Computational performance results for LOA inner iteration for PWR w/MOX 2g

#Processors	Runtime(s)	Speedup	Efficiency(%)
1	165	1.0	100
5	33	5.0	100
10	18	9.3	93
20	10	17	85
40	5	33	83

Table 15. Computational performance results for full-order inner iteration for PWR w/MOX 2g

#Processors	Runtime(s)	Speedup	Efficiency(%)
1	1010	1.0	100
5	217	4.7	94
10	127	8.0	80
20	65	16	80
40	36	28	70

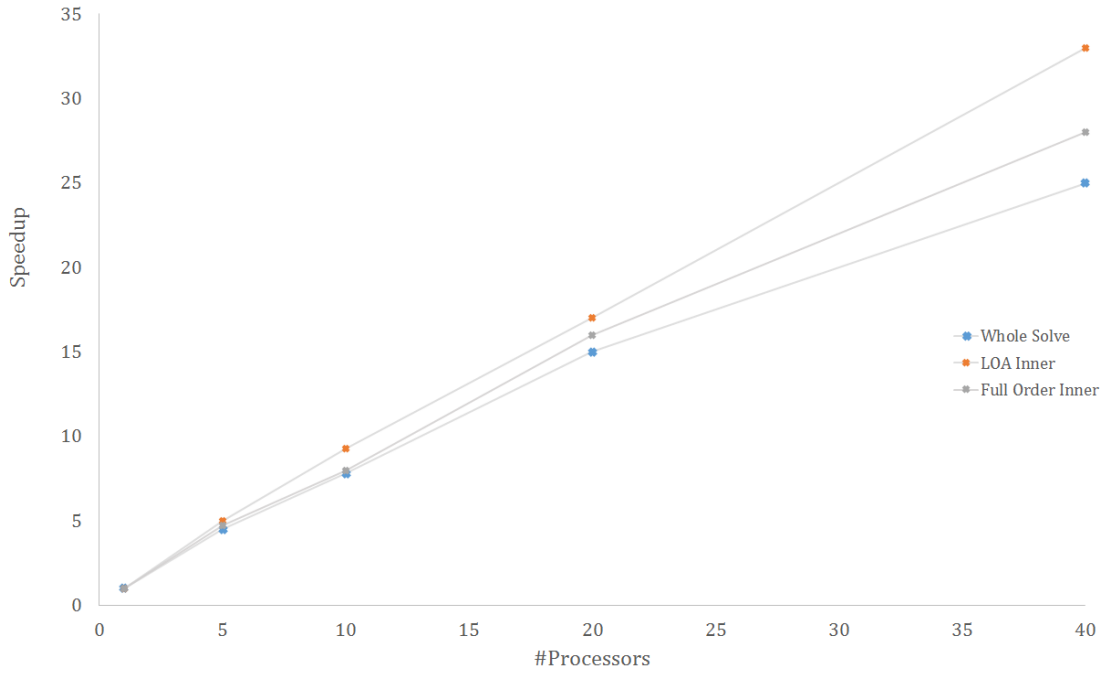

Figure 29. Speedups for the whole solve, LOA inner iteration, and full order inner iteration for the PWR with MOX 2 group problem.

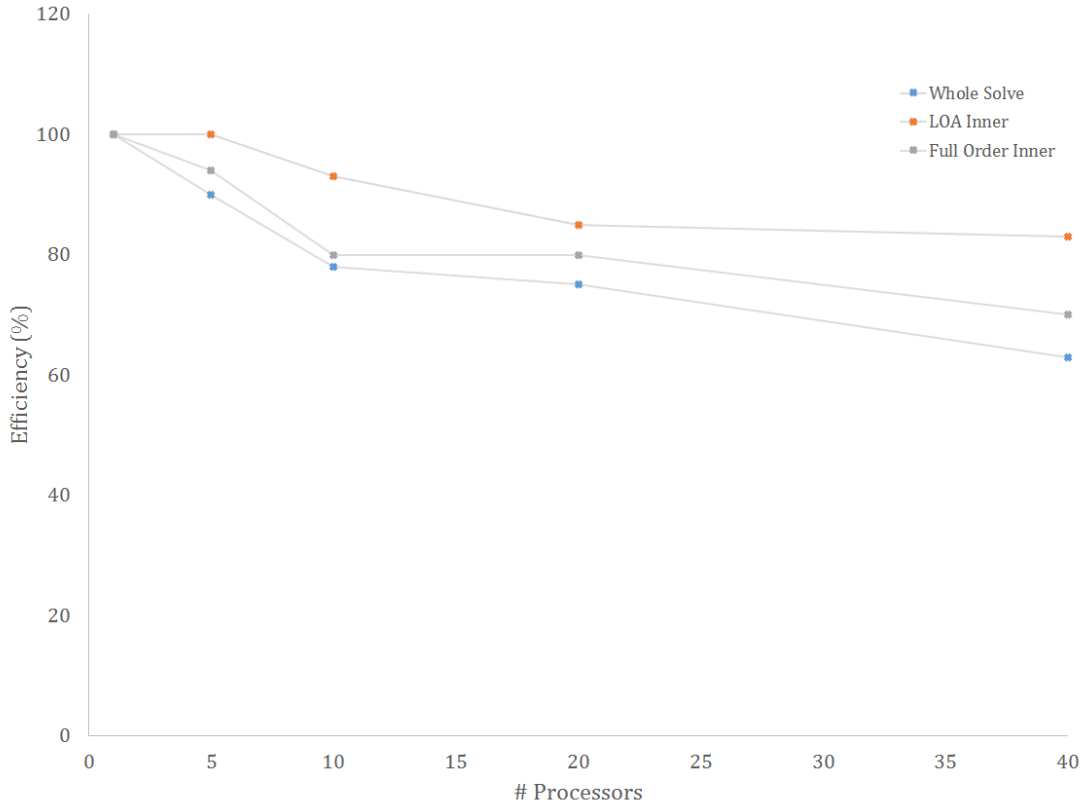

Figure 30. Parallel efficiencies for the whole solve, LOA inner iteration, and full order inner iteration for the PWR with MOX 2 group problem.

As with the PWR with gadolinium problem, here it is observed that on a small research cluster, a whole-core reactor problem is solved with COMET-MPI in merely *two minutes*, given in Table 13. Along with the previous problem, the flux expansion used is higher than previously employed in COMET solutions for this problem [3,8]. Figures 29 and 30 indicate near linear increase in speedup as number of processors increase and parallel efficiency behavior that is expected. In contrast to the previous problem, the parallel efficiency of the full order inner iteration is lower than the efficiency of the LOA inner iteration, indicating that the effect of response function lookups for this case is affecting the parallel efficiency.

Also in contrast to the previous case is that the parallel efficiency of the whole problem solution is less than either the LOA or full order inner solve. This is likely due to reading in responses from the response database during the outer iterations. While in the PWR with gadolinium problem there were many coarse meshes that could be read in, allowing for a high level of parallelizability, there were only nine unique coarse meshes in this model, limiting the parallelizability of that effect.

PWR with MOX – Eight Energy Groups

The whole core solutions presented thus far have produced encouraging results. However, solutions to problems with higher number of energy groups are often desirable. As such solutions for the PWR problem with MOX in eight energy groups were computed with COMET-MPI utilizing the full fourth order in space and second order in angle expansions. The calculated eigenvalue for these calculations was $k = 1.02011 \ +/- \ 0.5\, pcm$. The computational performance of the solutions for various numbers of processors in a calculation was recorded. Tables 16-18 show the computational performance for full core solve, the LOA inner iteration, and the full order inner iteration, respectively. As with the previous cases, the wall clock times for Table 16 are given in minutes, and the wall clock times for Tables 17-18 are given in seconds. The speedups and parallel efficiencies of the whole solve, LOA inner iteration, and full order inner iteration are plotted in Figures 31 and 32.

Table 16. Computational performance results for the full core PWR w/MOX 8g solution

#Processors	Wall Clock Time (min)	Speedup	Efficiency(%)
1	157	1.0	100
5	38	4.2	80
10	24	6.7	67
20	12	13	65
40	6.7	24	60

Table 17. Computational performance results for LOA inner iteration for PWR w/MOX 8g

#Processors	Wall Clock Time (s)	Speedup	Efficiency(%)
1	943	1.0	100
5	193	4.9	98
10	112	8.4	84
20	58	16	80
40	32	29	73

Table 18. Computational performance results for full-order inner iteration for PWR w/MOX 8g

#Processors	Wall Clock Time (s)	Speedup	Efficiency(%)
1	2347	1.0	100
5	596	3.9	78
10	393	6.0	60
20	196	12	60
40	100	23	58

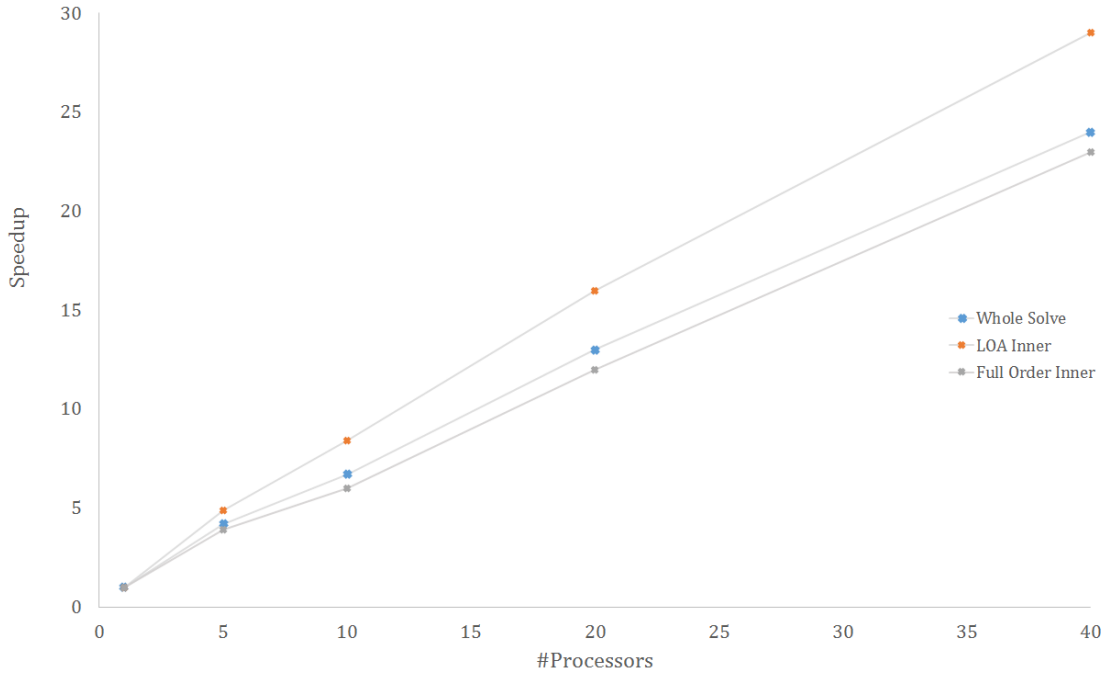

Figure 31. Speedups for the whole solve, LOA inner iteration, and full order inner iteration for the PWR with MOX 8 group problem.

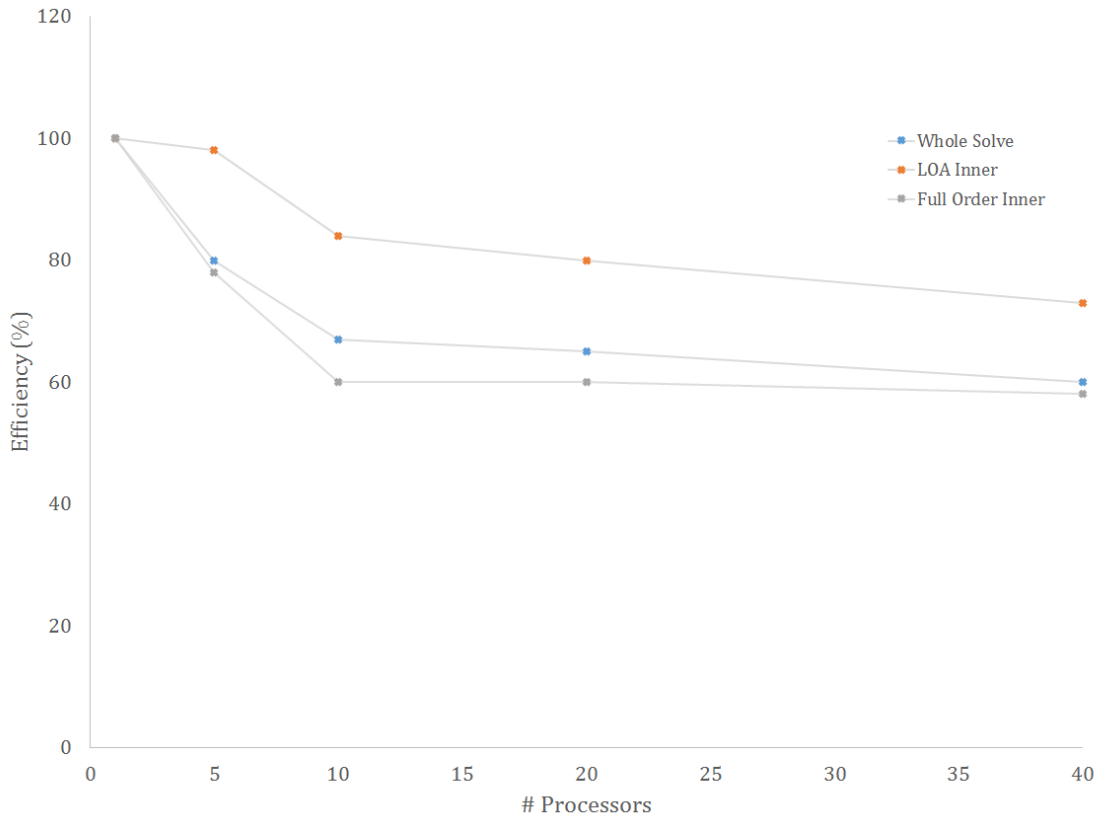

Figure 32. Parallel efficiencies for whole solve, LOA inner iteration, and full order inner iteration for the PWR with MOX 8 group problem.

Again, the computational performance is encouraging. When solving a problem with eight energy groups, the whole-core solve time with COMET-MPI is just under *seven minutes*, as seen in Table 16. For this benchmark problem, it is observed that the efficiency of the entire execution time follows very closely the execution time of the full order inner iteration. This is because this part of the program execution dominates for this case. Figures 31 and 32 indicate that as with the previous problems, speedup increases linearly with greater number of processors that and parallel efficiency behaves as expected.

7.3 Computational Performance with More Processors

The results shown thus far have all been computed on a small research cluster whose maximum processor load is 40. However, it is often desirable to run parallel programs with more processors. In the case of evaluating COMET-MPI, it would be of interest to assess the performance limits of the code on a larger number of processors than available on the research cluster. Some results have been generated on the Prometforce-6 cluster on the PACE clusters at Georgia Institute of Technology [33]. This cluster is a shared heterogeneous cluster by several computational research groups. Because this is a heterogeneous cluster with a shared scheduler, the computational performance results are less reliable, as different processors and connections between nodes will lead to different wall clock times. In addition, running codes on this cluster takes time, since jobs must wait in a queue. Nonetheless, some results have been generated.

Using the PWR with gadolinium and PWR with MOX (two energy group) computational models with maximum flux expansions in their respective models, computational performance results have been generated on Prometforce-6 for the case with fixed number of iterations – one outer iteration and 150 inner iterations, recording the runtime data for only the inner iterations, as was the case in the previous chapter's results. While core eigenvalues were computed for these runs, they are not reported since they are unimportant to the focus of this work. Computational wall clock times for two trials of runs for the PWR with gadolinium problem were recorded and are presented in Tables 19-20. Performance results for the PWR with MOX problem are presented in Table 21. The speedups and parallel

efficiencies for the PWR with gad runs are plotted in Figures 33 and 34. The speedup and parallel efficiency of the PWR with MOX run are plotted in Figures 35 and 36.

Table 19. Computational performance results for PWR w/gad on Prometforce-6, first run.

#Processors	Wall Clock Time(s)	Speedup	Efficiency(%)
1	1091	1.0	100
5	450	2.4	48
10	243	4.5	45
20	152	7.2	36
40	69	16	40
60	65	17	28
80	50	22	28
100	41	27	27
125	29	38	30

Table 20. Computational performance results for PWR w/gad on Prometforce-6, second run.

#Processors	Wall Clock Time (s)	Speedup	Efficiency(%)
1	1437	1.0	100
5	2064	0.7	14
10	260	5.5	55
20	166	8.7	44
40	188	7.6	19
60	65	22	37
80	57	25	31
100	48	30	30
125	59	24	19

Table 21. Computational performance results for PWR w/MOX on Prometforce-6.

#Processors	Wall Clock Time (s)	Speedup	Efficiency(%)
1	956	1.0	100
5	172	5.6	112
10	90	10.6	106
20	73	13	65
40	32	30	75
60	22	43	72
80	37	26	33
100	19	50	50

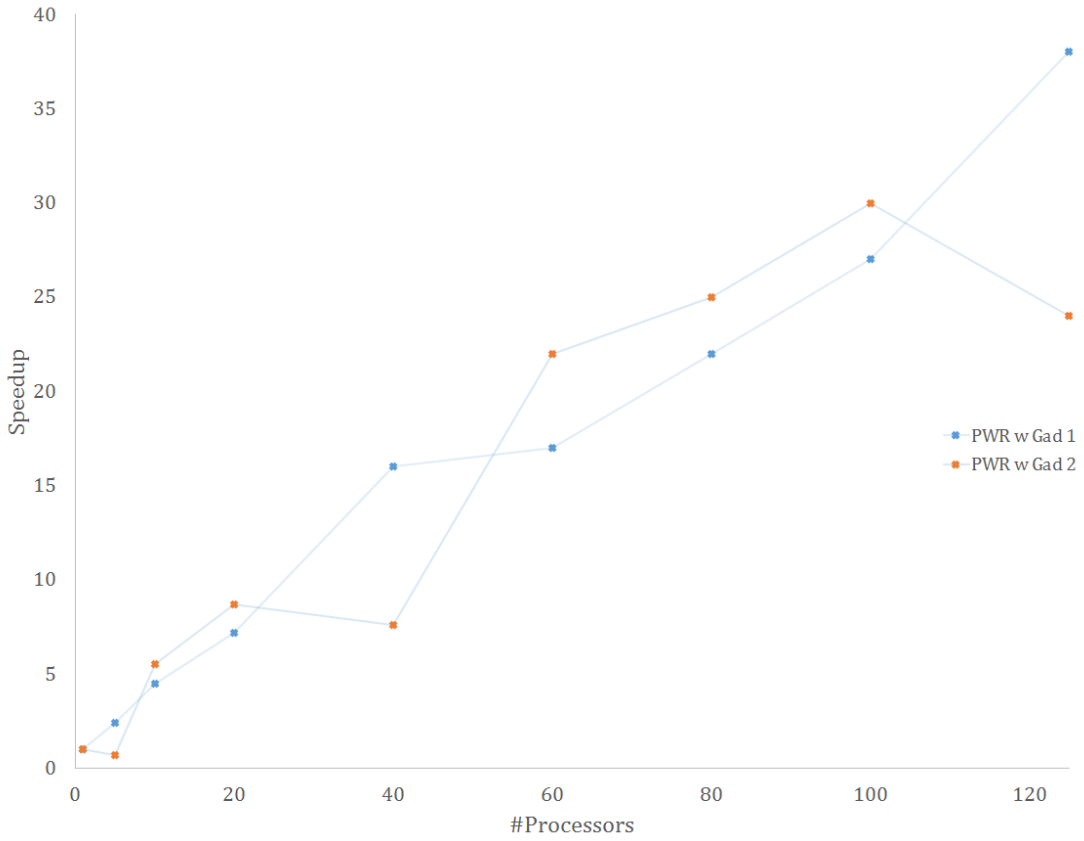

Figure 33. Speedups for both PWR with gad runs on Prometforce-6.

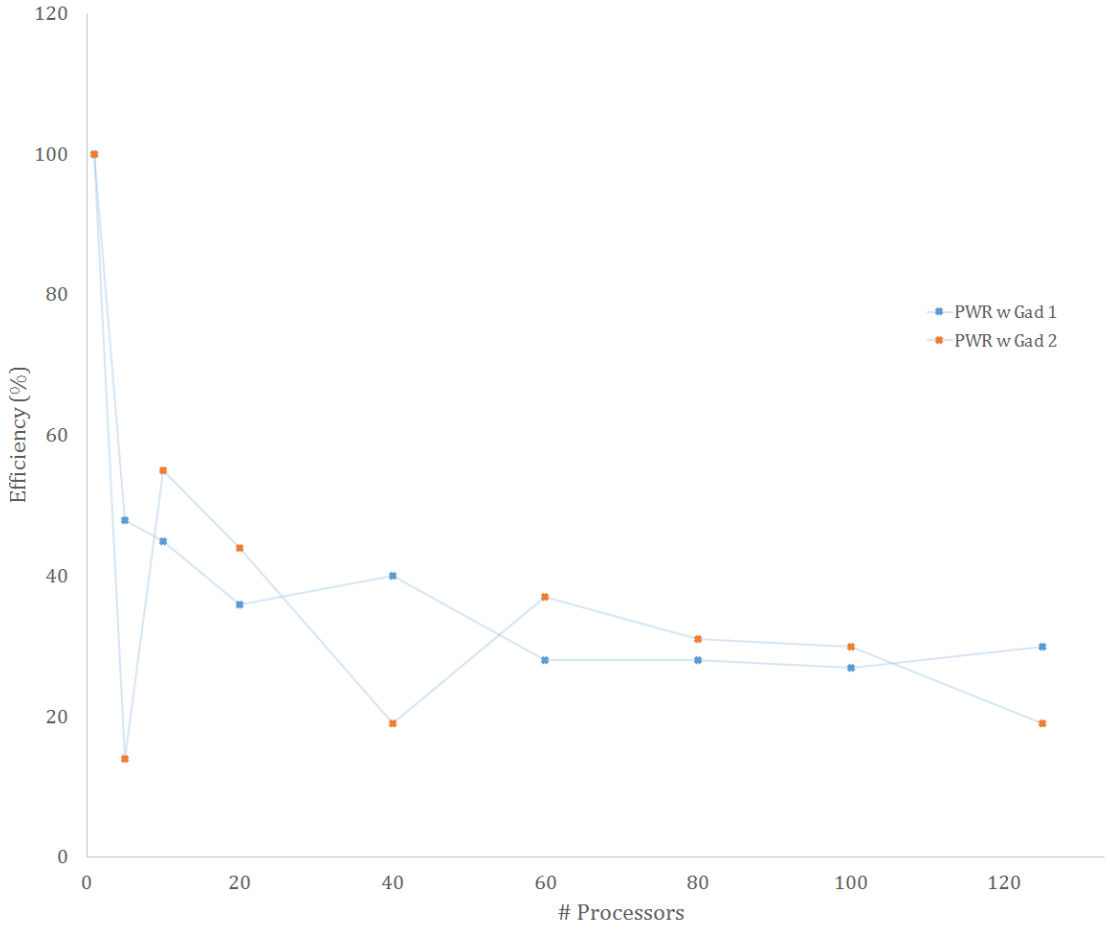

Figure 34. Parallel efficiencies for both PWR with gad runs on Prometforce-6.

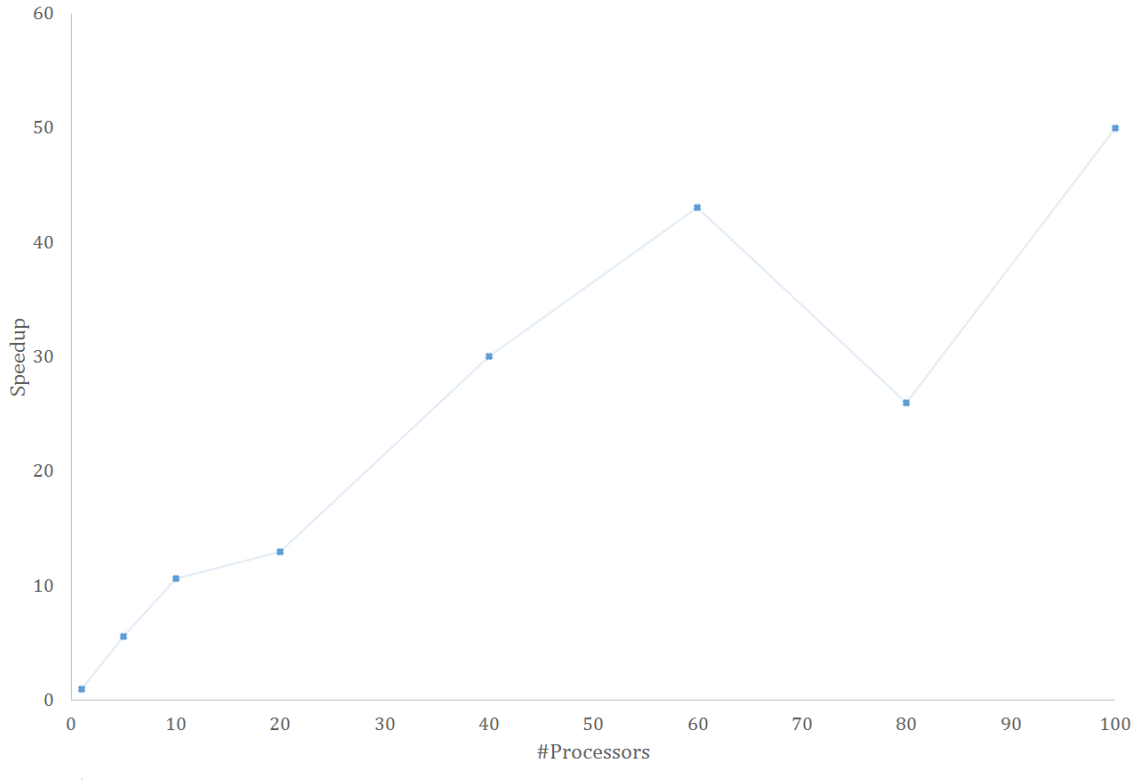

Figure 35. Speedup for the PWR with MOX runs on Prometforce-6.

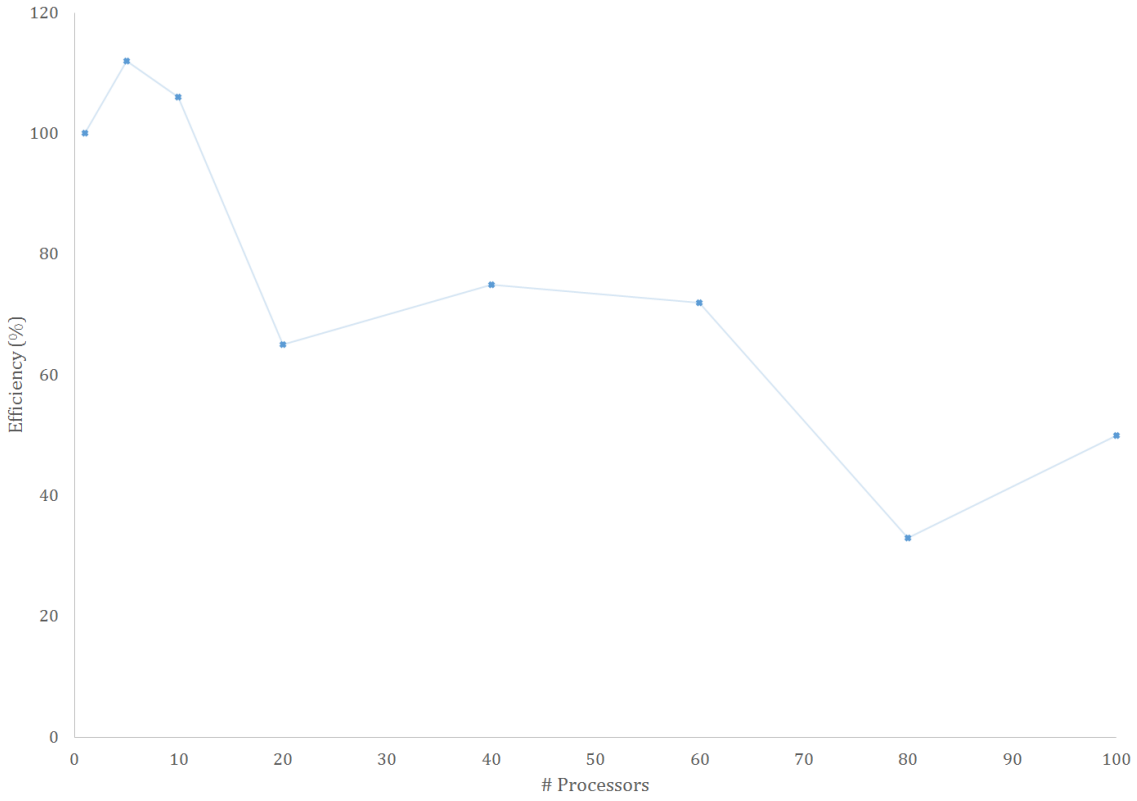

Figure 36. Parallel efficiencies for the PWR with MOX runs on Prometforce-6.

The results of these calculations with COMET-MPI confirm what has been stated previously; these computations were carried out on a heterogeneous cluster with a shared scheduler, so the absolute computational performance of the code will be difficult to assess on this cluster. As expected, the performance results are inconsistent, with some runs taking *more* time on more processors rather than less. Regardless, the trend of increasing processors used in a calculation leads to a trend of reduced runtime (up to a factor of 50). In practice, to perform calculations in the future, COMET-MPI can be used on a heterogeneous cluster such at Prometforce-6; however, the absolute performance characteristics of the code cannot be accurately observed.

The main goal of this project was to achieve efficient whole-core reactor calculations through the use of parallel computing. These efficient calculations have been demonstrated on a few PWR benchmark problems that are representative of the class of problems that have been encountered in COMET calculations. The more efficient solution of whole-core problems enables COMET to be coupled to other modules, such as depletion, thermal hydraulics, and on-the-fly response generation. With the exception of response generation, neutronics solves are typically the bottleneck in coupled calculations. With the computational efficiency shown in this chapter, this may no longer be the case.

CHAPTER 8

CONSIDERATIONS FOR PROBLEMS WITH HIGH FLUX EXPANSIONS

The results presented thus far have been very encouraging. On just a small research cluster, increases in computational efficiency as high as a factor of 35 have been observed. In addition to simply improving efficiency for the types of calculations that COMET has encountered in the past, it is tempting to determine what types of new problems are now computationally tractable. As the sensitivity studies in Chapter 6 indicate, problems with increasing numbers of coarse meshes might be more easily tackled with the use of parallel computing. However, the sensitivity studies also indicate that for high enough flux expansions, parallel efficiency might degrade due to the expense of response function lookups in the inner iteration. This effect is investigated in this chapter on (again) the PWR with MOX benchmark problem.

Using an eight energy group library, a response library with maximum flux expansions of fourth order in space *and* angle was compiled, again using the code MCNP5. The responses tallied were the same as in previous computational models of this problem. COMET-MPI was used to compute solutions to this problem. However, iterations were limited to one outer iteration and 150 inner iterations, and computational runtime data was recorded only for the inner iterations. Still, in this case, the problem poses *severe* difficulty for the serial case, as even just one

set of inner iterations takes several hours to calculate. As before, COMET-MPI calculations took place on a small research cluster of Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz processors, with 2 nodes of 20 processors each. While core eigenvalues were computed for these runs, they are not reported since they are not important to the focus of this work. The wall clock time results for COMET-MPI for various processors is given in Table 22. Wall clock times are reported in hours.

Table 22. Computational performance results for full-order inner iteration for PWR w/MOX 8g with high flux expansion.

#Processors	Wall Clock Time (h)	Speedup	Efficiency(%)
1	4.6	1.0	100
5	2.4	1.9	38
10	2.2	2.1	21
20	1.0	4.6	23
40	N/A	N/A	N/A

As Table 22 indicates, performance of COMET-MPI for this high flux expansion case is poor. The code did not produce a result for the 40 processor run because the memory requirements exceeded the resources available on the research cluster. While the use of COMET-MPI reduced the runtime of the calculation from a few hours to one hour, the parallel efficiency for the problem is low.

These degradations in computational efficiency are amplified examples of deviations from the scalability model in chapter 4 due to the costs of response function lookups in the inner iteration. For problems that use high amounts of

memory for the response matrices (as is the case for problems with high flux expansions and high numbers of energy groups), these memory operations become quite formidable. In addition, the requirement of an MPI application that each processor have its own local memory can be prohibitive for cases such as this problem where memory requirements are rather high.

A potential remedy for this is a hybrid software implementation of MPI *and* OpenMP [34]. In these types of software implementations, a problem is decomposed as it would be in a distributed memory MPI-only approach, but then each problem chunk is further parallelized with OpenMP. An advantage of this implementation is that it limits memory requirements, as the shared-memory aspect of OpenMP can be utilized.

This approach was briefly investigated in this study. While the implementation explored is hardly exhaustive, the coarse mesh sweeps were multithreaded using the OpenMP compiler directive *!$omp parallel do*. Runs using the hybrid MPI+OpenMP approach were carried out, and the performance results are given in Table 23. Wall clock times are again reported in hours.

Table 23. Computational performance results for full-order inner iteration for PWR w/MOX 8g with high flux expansion, MPI+OpenMP.

#Processors	Wall Clock Time (hour)	Speedup	Efficiency(%)
1	4.6	1.0	100
5	1.1	4.4	88
10	0.88	5.2	52
20	0.68	6.7	34
40	0.81	5.7	14

In the runs shown in Table 23, for the cases up to 20 processors, the parallelism was entirely through multithreading in OpenMP. For the 40 processor case, the problem was decomposed onto to distributed memory processes, and then each of these processes utilized 20 threads. While the efficiency for 5 threads is much better than the case for 5 processors without OpenMP in COMET-MPI, the speedup saturates very quickly for this implementation. Further, even though a 40 processor run was able to be achieved in this implementation, it is slower than the 20 processor computation. This is because each calculation was only carried out on a single node, even though two were available. The scheduler used on the cluster always sent all processes/threads to one node rather than sharing the work. In general, in the future, it may be difficult to choose what jobs will be sent to which nodes if a scheduler is in charge of starting computing jobs.

Another issue of this implementation is permutations of MPI decompositions and OpenMP threads. Though in this case, OpenMP multithreading dominated the parallelism, it is possible to primarily decompose a problem with MPI and then utilize only a few threads. However, there are many different possible combinations, and determining the optimal combinations of decompositions and threads may be difficult to determine in a systematic way.

CHAPTER 9

CONCLUSIONS

In this thesis, a distributed memory algorithm was developed and implemented for use in the COMET method under the motivation of improving computational efficiency of calculations as well as paving the way for coupling to other software modules, such as multiphysics (e.g., depletion or thermal hydraulics) and on-the-fly response generation. A software implementation for this algorithm was developed, COMET-MPI. This proxy app was tested for various benchmark problems. In all cases, the use of multiprocessing via COMET-MPI improved the computational efficiency of COMET calculations. The project achieved the major goal of efficient whole-core simulations utilizing parallel computing.

The COMET-MPI code was benchmarked against expected speedups and parallel efficiencies predicted by Amdahl's Law. It was determined during the course of this study that in addition to calculation and inter-processor communication costs, the cost of response function lookups during the inner iteration greatly affects the computational performance of the code. An important aspect of future work will be to investigate ways to limit the effect of response function lookups on parallel efficiencies, thus increasing the speed of calculations and scalability of future implementation of COMET calculations employing parallel computing.

Sensitivity calculations that investigated the effect of problem size on parallel efficiency were carried out. It was observed that for problems with greater numbers of coarse meshes, parallel efficiency is improved, helped both by increased scalability of the code as well as potentially mitigating the effect of response function lookup costs. Building on this result, it would be an interesting focus of future work to perform calculations with tighter spatial meshing (e.g., more axial zones) to get finer spatial distributions of pin fission densities in reactor cores. Further, if smaller coarse meshes are used in response generation, the computational burden for this stage of calculations is lessened, and the permutations of problems that can be solved in the deterministic algorithm given a response library is greatly increased.

COMET-MPI was used to solve whole-core benchmark problems indicative of the types of problems of past and future research interest. In this case, COMET-MPI was able to compute solutions to these traditionally challenging problems in just *minutes*. These extremely efficient calculations are highly encouraging, as a major goal of the project was increasing efficiency of whole-core calculations.

A problem with a higher flux expansion than usually used in COMET calculations was also explored with COMET-MPI. While performance for this calculation was improved with the use of multiprocessing, the parallel efficiency for this case was poor; response function lookups during the inner iteration proved to be a major bottleneck in the problem execution. A hybrid MPI+OpenMP approach was explored to remedy this, but the results were inconclusive. An obvious path of future work, then, is to look for ways to improve parallel efficiency

for the high flux expansion case, including more sophisticated implementations of MPI+OpenMP.

Another obvious area of future work is extending a parallel implementation of COMET to many more processors. While computational resources were somewhat limited during the course of work of this thesis, it would be interesting to see how the COMET-MPI code or a future implementation behaves with more processors used to solve a problem. To this end, hybrid decomposition strategies (e.g., domain *and* flux expansion decomposition) should be explored to allow for decomposition on the order of tens of thousands of processors to enable *massively* parallel computations. An important aspect of this future work is an update in the response function library data storage method. CDF libraries are currently used, but this affects portability of the code to different machines, as compilation and execution of the code becomes more complicated.

Overall, the development of the proxy app COMET-MPI demonstrated improvements in computational efficiency and data handling over previous serial implementations of the COMET code. As COMET calculations are coupled to on-the-fly response generation and mulitphysics modules, it is recommended that a parallel implementation of the COMET code is used.

APPENDIX A

TABULATED COMPUTATIONAL PERFORMANCE DATA FOR AMDAHL'S LAW STUDIES

The tabulated computational performance data for the Amdahl's Law studies of Chapter 5 are provided in Tables A1-A4 below.

Table A1. Computational performance for the C5G7 problem, response function lookup case.

#Processors	Wall Clock Time (s)	Speedup	Efficiency (%)
1	103.0	1.0	100.0
2	61.6	1.7	83.6
3	52.0	2.0	66.1
4	42.4	2.4	60.8
5	26.1	3.9	78.8
6	24.4	4.2	70.4
7	22.5	4.6	65.5
8	21.7	4.8	59.4
9	20.9	4.9	54.9
10	21.3	4.8	48.3
11	19.3	5.3	48.5
12	18.5	5.6	46.3
13	17.0	6.1	46.7
14	16.2	6.4	45.6
15	15.1	6.8	45.6
16	14.2	7.3	45.3
17	13.3	7.7	45.5
18	12.5	8.3	45.9
19	12.2	8.4	44.3
20	12.0	8.6	42.9
21	11.6	8.9	42.4
22	11.1	9.3	42.3
23	11.0	9.4	40.9
24	10.4	9.9	41.4
25	10.1	10.2	40.7
26	9.5	10.9	41.9

27	9.4	10.9	40.6
28	9.0	11.5	41.0
29	9.0	11.4	39.4
30	9.2	11.3	37.5
31	8.6	12.0	38.8
32	7.9	13.0	40.5
33	8.0	12.8	38.8
34	7.7	13.5	39.6
35	6.7	15.4	44.1
36	6.5	15.7	43.7
37	6.5	15.8	42.7
38	6.7	15.3	40.2
39	6.9	15.0	38.5
40	7.0	14.6	36.6

Table A2. Computational performance for the C5G7 problem, no response function lookup case.

#Processors	Wall Clock Time (s)	Speedup	Efficiency (%)
1	100.2	1.0	100.0
2	50.9	2.0	98.4
3	34.2	2.9	97.6
4	25.9	3.9	96.7
5	20.8	4.8	96.2
6	17.2	5.8	97.0
7	14.9	6.7	95.9
8	13.1	7.6	95.6
9	11.8	8.5	94.5
10	11.3	8.9	88.8
11	9.9	10.1	92.1
12	9.0	11.2	92.9
13	8.4	12.0	92.0
14	7.8	12.8	91.7
15	7.3	13.7	91.1
16	6.7	14.9	93.1
17	6.4	15.8	92.7
18	6.0	16.8	93.4
19	5.9	17.1	90.1
20	5.6	17.8	89.2
21	5.2	19.1	91.0
22	5.1	19.6	89.2

23	4.8	20.7	89.8
24	4.5	22.2	92.6
25	4.6	22.0	88.0
26	4.4	23.0	88.3
27	4.1	24.4	90.4
28	4.1	24.7	88.3
29	4.1	24.2	83.3
30	3.7	26.8	89.2
31	3.8	26.2	84.4
32	3.4	29.6	92.4
33	3.4	29.5	89.4
34	3.5	28.3	83.2
35	3.3	30.7	87.7
36	3.0	33.0	91.8
37	3.1	32.7	88.5
38	3.0	33.1	87.2
39	3.0	33.5	86.0
40	3.0	33.6	83.9

Table A3. Amdahl speedups and efficiencies, response function lookup case.

#Processors	Amdahl Speedup	Ahmdahl Efficiency (%)
1	1.0	99.4
2	1.9	93.2
3	2.6	87.8
4	3.3	83.0
5	3.9	78.7
6	4.5	74.8
7	5.0	71.3
8	5.4	68.0
9	5.9	65.1
10	6.2	62.4
11	6.6	59.9
12	6.9	57.7
13	7.2	55.5
14	7.5	53.6
15	7.8	51.7
16	8.0	50.0
17	8.2	48.4
18	8.4	46.9
19	8.6	45.5
20	8.8	44.2

21	9.0	42.9
22	9.2	41.7
23	9.3	40.6
24	9.5	39.5
25	9.6	38.5
26	9.8	37.6
27	9.9	36.7
28	10.0	35.8
29	10.1	35.0
30	10.3	34.2
31	10.4	33.4
32	10.5	32.7
33	10.6	32.0
34	10.7	31.3
35	10.7	30.7
36	10.8	30.1
37	10.9	29.5
38	11.0	28.9
39	11.1	28.4
40	11.1	27.9

Table A4. Amdahl speedups and efficiencies, no response function lookup case.

#Processors	Amdahl Speedup	Ahmdahl Efficiency (%)
1	1.0	99.4
2	2.0	98.8
3	2.9	98.2
4	3.9	97.6
5	4.8	97.0
6	5.8	96.4
7	6.7	95.8
8	7.6	95.3
9	8.5	94.7
10	9.4	94.2
11	10.3	93.6
12	11.2	93.1
13	12.0	92.5
14	12.9	92.0
15	13.7	91.5
16	14.6	91.0
17	15.4	90.5

18	16.2	90.0
19	17.0	89.5
20	17.8	89.0
21	18.6	88.5
22	19.4	88.0
23	20.1	87.5
24	20.9	87.0
25	21.6	86.6
26	22.4	86.1
27	23.1	85.7
28	23.9	85.2
29	24.6	84.8
30	25.3	84.3
31	26.0	83.9
32	26.7	83.4
33	27.4	83.0
34	28.1	82.6
35	28.8	82.2
36	29.4	81.8
37	30.1	81.3
38	30.8	80.9
39	31.4	80.5
40	32.1	80.1

REFERENCES

[1] Mosher, S., Rahnema, F., "The Incident Flux Response Expansion Method for Heterogeneous Coarse Mesh Transport Problems," Transport Theory and Statistical Physics, vol. 35(1), pp. 55-96, 2006.

[2] Zhang, D., Rahnema, F., "An Efficient Stochastic/Deterministic Coarse Mesh Neutron Transport Method," Annals of Nuclear Energy, vol. 41(1), pp. 1-11, 2012.

[3] Rahnema, F., Zhang, D., Connolly, K., "The COMET method for reactor physics calculations," PBNC2014-099, Presented at the Pacific Basin Nuclear Conference, Vancouver, British Columbia, Canada, August 24-28 2014.

[4] Ulmer, R., Rahnema, F., Connolly, K.J., "A Neutronic Benchmark Specification and COMET Solution for the Advanced Burner Test Reactor," Ann. Nucl. Energy, accepted, July (2015).

[5] Connolly, K.J., Rahnema, F., "Heterogeneous Coarse Mesh Radiation Transport Method for Neutronic Analysis of Prismatic Reactors," Ann. Nucl. Energy, 56, 87-101 (2013).

[6] Remley, K., Rahnema, F., "Fuel pin problem solution via an accelerated COMET method," Transactions of the American Nuclear Society, Presented at the American Nuclear Society Winter Meeting, Anaheim, CA, November 9-13 2014.

[7] Remley, K., Rahnema, F., "An adaptive COMET solution to a configuration of the C5G7 benchmark problem," ANS MC2015 - Joint International Conference

on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Presented at ANS MC2015, Nashville, TN, April 19-23 2015.

[8] K. Remley and F. Rahnema, “A Method for the Adaptive Selection of Angular Flux Expansion Orders in the Coarse Mesh Radiation Transport (COMET) Method,” Nuclear Science and Engineering, Volume 183(2) 161-173.

[9] K. Remley et al. “Application of an Adaptive COMET Method to a PWR Benchmark Problem with Gadolinium,” Nuclear Science and Techniques, pending, June (2016).

[10] Lago, D. “Development of an Application Programming Interface for Depletion Analysis (APIDA).” Ph.D. Thesis. Nuclear and Radiological Engineering/Medical Physics. Atlanta, GA. Georgia Institute of Technology. May 2016.

[11] Hon, R. “Development of Methods and Tools for On-the-Fly Response Function Generation for Criticality Calculations.” Ph.D. Thesis. Nuclear and Radiological Engineering/Medical Physics. Atlanta, GA. Georgia Institute of Technology. May 2016.

[12] Lewis, E. E., Miller, W. F., Jr. (1984). Computational Methods of Neutron Transport. La Grange Park, IL: American Nuclear Society.

[13] R.N. Slaybaugh, T.M. Evans, G.G. Davidson, and P.P.H. Wilson, Rayleigh Quotient Iteration with a Multigrid in Energy Preconditioner for Massively Parallel Neutron Transport," Proceedings of Joint International Conference on

Mathematics and Computation, Supercomputing in Nuclear Applications, and the Monte Carlo Method in Nashville, TN, April 2015.

[14] G. Sjoden and A. Haghighat, “PENTRAN: A parallel 3-D S(N) transport code with complete phase space decomposition, adaptive differencing, and iterative solution methods,” Pennsylvania State University, 1997.

[15]A. Marin-Lafleche, M. A. Smith, and Chang-ho Lee, “PROTEUS-MOC: A 3D Deterministic Solver Incorporating the 2D Method of Characteristics,” International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, Sun Valley, Idaho, May 5-9, 2013.

[16] R. Hayward and F. Rahnema, “COMET-PE: An Incident Fluence Response Expansion Transport Method for Radiotherapy Calculations,” Physics in Medicine and Biology, Volume 58(1), pp. 3125-3143 (2013).

[17] J. Roberts and B. Forget, “Solving Eigenvalue Response Matrix Equations with Nonlinear Techniques,” Ann. Nucl. Energy, 69, 97-107 (2014).

[18] Kelley, C.T., 1995. Iterative Methods for Linear and Nonlinear Equations. Society for Industrial Mathematics.

[19] Stamm’ler, R.J.J., Abbate, M.J., 1983. Methods of Steady-state Reactor Physics in Nuclear Design. Academic Press, London-New York.

[20] Flynn, M., "Some Computer Organizations and their Effectiveness," IEEE Trans Comput, C-21:948-960 (1972).

[21] Sjoden, G., Haghighat, A., 2008. PENTRAN™ Code System Parallel Environment Neutral-Particle TRANsport Parallel Distributed Decomposition

of Discrete Ordinates (Sn) in 3-D Cartesian Geometry Users Guide to Version 9.4X.5 Series. HSW Technologies.

[22] W. Gropp, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing, The MIT Press, Cambridge, Massachusetts, London, England, 1999.

[23] Brantley, P., Hamilton, S., Kunen, A., Romano, P., Zerr, J., "Computational Methods – Roundtable: Proxy Apps, Mini Apps, and Research Apps: Progress and Lessons Learned," Transactions of the American Nuclear Society, Presented at the American Nuclear Society Annual Meeting, New Orleans, LA, June 12-16 2016.

[24] X-5 Monte Carlo Team, MCNP – A General N-Particle Transport Code, Version 5, Los Alamos National Laboratory, Los Alamos, New Mexico, 2005.

[25] Roberts, J.A., 2014. Advanced Response Matrix Methods for Full Core Analysis, Ph.D. Thesis, Massachusetts Institute of Technology.

[26] Forget, B., 2006. A Three Dimensional Heterogeneous Coarse Mesh Transport Method for Reactor Calculations, Ph.D. Thesis, Georgia Institute of Technology.

[27] NSSDC: National Space Science Data Center (2016), "Common Data Format (CDF)," http://cdf.gsfc.nasa.gov/, last accessed on May 29th, 2016.

[28] Smith, M., Lewis, E., Na, B., Benchmark on Deterministic Transport Calculations Without Spatial Homogenization: MOX Fuel Assembly 3-D Extension Case, OECD/NEA Report NEA/NSC/DOC, Paris, France, 2005.

[29] V. Kucukboyaci, A. Haghighat, and G.E. Sjoden, "Performance of PENTRAN™ 3-D Parallel Particle Transport Code," Euro PVM/MPI 2001, LNCS 2131, pp. 36–43, 2001

[30] M. Horoi and R. Enbody, "Using Amdahl's Law as a Metric to Drive Code Parallelization: Two Case Studies," The International Journal of High Performance Computing Applications Volume 15, No. 1, Spring 2001, pp. 36-41

[31] S. Douglass, F. Rahnema, and J. Margulies, "A stylized three dimensional PWR whole-core benchmark problem with gadolinium," *Annals of Nuclear Energy*, Vol. 27, 2010, pp. 1384-1403.

[32] Rahnema, F. Hon, R., Douglass, S., "A Stylized Three-Dimensional Pressurized Water Reactor Benchmark Problem with UO_2 and MOX Fuel," Nuclear Technology, vol. 183 pp. 1-28 (2013).

[33] Partnership for an Advanced Computing Environment (PACE), "Overview – Partnership for an Advanced Computing Environment (PACE)," http://pace.gatech.edu/overview, last accessed on May 29th, 2016.

[34] Rabenseifner, R., Hager, G., & Jost, G. (2009, February). Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In 2009 17th Euromicro international conference on parallel, distributed and network-based processing (pp. 427-436). IEEE.

VITA

Kyle Eugene Remley

Kyle Remley was born in Greenville, South Carolina in 1991, on January 1st, in a brilliantly calculated move to ensure no one would ever forget his birthday. Kyle graduated from Berkmar High School in Lilburn, GA in May of 2009. Following this, he began his lengthy tenure at the Georgia Institute of Technology. Kyle earned his B.S in Nuclear and Radiological Engineering in the May of 2013. Deciding Tech was too much fun to quit, he remained there for graduate school. During this time, he earned his M.S. in Nuclear Engineering in May of 2015. Kyle currently lives in Snellville. When he is not doing research, he enjoys playing drums in a rock band, collecting music CDs, running, and annoying his labmates with unsolicited long-winded rants about the band Rush.