A Hybrid Parallel Algorithm for the 3-D Method of Characteristics Solution of the Boltzmann Transport Equation on High Performance Compute Clusters

by

Brendan Matthew Kochunas

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Nuclear Engineering and Radiological Sciences)
in the University of Michigan
2013

Doctoral Committee:

Professor Thomas J. Downar, Chair
Professor Edward W. Larsen
Professor William R. Martin
Scott Palmtag, Core Physics Inc.
Professor Quentin F. Stout

For my Dad, Brad Kochunas

For teaching me the value and joy of intellectual pursuits

ACKNOWLEDGEMENTS

This work would not have been possible without the tireless efforts of my advisor, Prof. Tom Downar. I followed him across the country and back, and have had the good fortune (over the past 8 years) to have had invaluable learning experiences and inimitable opportunities every step of the way. I also want to acknowledge the other members of my PhD committee, and in particular Prof. Ed Larsen for their support. Each has contributed in some way to my own insights. I would like to thank my first mentor, Dr. Justin Thomas, for getting me started on the right foot, as well as several other former students of Prof. Downar with whom I've had the pleasure of working. In particular, I want to also thank Dr. Mathieu Hursin for being an exceptional colleague and mentor. I also owe thanks to Zhouyu Liu, whose arrival could not have been more timely.

I feel I am also incredibly indebted to the original authors of the DeCART code: Prof. Han-Gyu Joo, Dr. Jin-Young Cho, and Dr. Kang-Seog Kim. Although I did not personally work with any of them, they deserve a very special acknowledgement for having designed a code that, to me, was an exceptional teaching tool.

I must also acknowledge the CASL program for its support of this work and for providing the necessary resources for enabling its completion. My colleagues at the University of Michigan who also contribute to MPACT deserve to be acknowledged, as they kept the final leg of my journey interesting to say the least.

I would also like to thank my family (Brad, Marge, and Abbi) for all they have done (or deny doing) to raise me. Certainly this would not have been possible without them. Finally, I would also like to acknowledge the distinguished gentlemen of 144 Hill Street, without whom this whole experience would have been far more tiresome.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

The focus of this thesis is on the development of a highly scalable parallel algorithm for solving the 3-D method of characteristics (MOC) form of the Boltzmann neutron transport equation. The derivation of the 3-D MOC method is presented first, along with the details of the discretization techniques, that utilize the concept of modular ray tracing. The implementation of these equations is then described, and then the approach to parallelizing the algorithm is discussed. Results are shown for a range of benchmark problems typically solved by 3-D neutron transport codes.

The algorithm is parallelized in space, angle, and by characteristic rays, which is specific to the MOC solution method. Once the parallel algorithm is established, a performance model for the particular implementation is derived. This model contains detailed expressions for the number of floating point operations and execution time as a function of the problem size and fundamental computer hardware properties, such as the time per flop and cache access latency.

The procedure for determining the hardware coefficients required by the performance model is then presented and validated using experimental results. The performance model is shown to agree well with experiment for both types of execution, and the model is therefore used for subsequent analyses that explore the algorithm's sensitivities to the computer and network hardware characteristics. The model is also analyzed to assess the scaling of the algorithm for a quarter core PWR.

The optimization of the convergence of the parallel 3-D MOC algorithm through the use of the coarse mesh finite difference (CMFD) method is then developed. The CMFD accelerated parallel 3-D MOC algorithm is then used to compute solutions to several numerical benchmarks, that show good agreement with the reference results. Finally, the research performed in this thesis and its conclusions are summarized, and areas of future research are suggested.

Chapter 1

INTRODUCTION

The field of nuclear reactor physics has matured considerably since it began in the 1940's, and the use of computer simulations of reactor core neutronics behavior was adopted very early on [1]. Reactor physicists have been reasonably successful in predicting the behavior of a reactor for a wide range of both steady-state and transient conditions. Historically, the methods used for reactor simulation have largely been determined by the computational resources available at the time. A historical survey of the literature shows an evolution from the four and six factor formulas to two-group and few-group neutron diffusion theory for practical LWR core calculations, and today, two-group neutron diffusion theory continues to be the workhorse for practical LWR core calculations. A principal focus of much of the research over the years was to improve methods for generating homogenized few-group cross sections for the core calculation. It has only been recently with the availability of petascale computing that LWR researchers have focused considerable efforts on performing whole-core LWR calculations using higher order transport methods [2], [3], [4], [5], [6], [7], [8], [9], [10].

High performance computing technology and the availability of petascale machines have made it possible to perform full core transport calculations in a reasonable amount of time. A simple estimate is shown below of the computational requirements for a practical LWR whole-core transport solution. This estimate assumes typical discretizations of the space, energy, and angular dependent neutron flux and solution times per unknown currently seen in routine assembly cross section calculations.

Table 1.1 – Estimated computational requirements for a deterministic whole-core transport calculation

Number of Neutron Energy Groups	50
Number of Discrete Angles	128
Number of Spatial regions	1,140,528,096
Number of Unknowns (Angular Flux)	7,299,379,814,400
Average MFLOPS to Solve each Unknown	4
Estimated Memory Requirements (for only Angular Flux)	53.11 TB
Estimated Run time (for 1 petflop machine)	8 hrs

The principal economic motivation for improved reactor core calculations has been to increase reactor power density and operational flexibility without compromising reactor safety. Higher fidelity core calculations make it possible to relax the conservatism in safety margins and provide the tools for better understanding the full physics of current operational constraints. In fact, this has been the mission of a recent significant DOE research program, the Consortium for the Advanced Simulation of Light Water Reactors (CASL) [10], which has focused on advancing the modeling and simulation of LWR technology. The work described in this thesis was supported through this program.

1.1 Fundamental Challenges of High Fidelity Light Water Reactor Simulations

Perhaps the first challenge in high fidelity LWR simulations is accurately representing the geometric complexity of many of the reactor components. Some of these are shown for a typical PWR in Figure 1.1, taken from a recent paper [11] describing these issues in detail. To be noted in this figure are such components as the upper plenum region of a fuel rod, spacer grids, the bottom and top nozzle of the assembly, and the core shroud or baffle and the barrel. Traditionally, these components have not been explicitly modeled in reactor core calculation, and instead were simply homogenized with the fuel, cladding, water and other materials into the few group assembly cross sections. One of the principal innovations of the next generation of methods is to explicitly represent each of these components and materials during the core calculation.

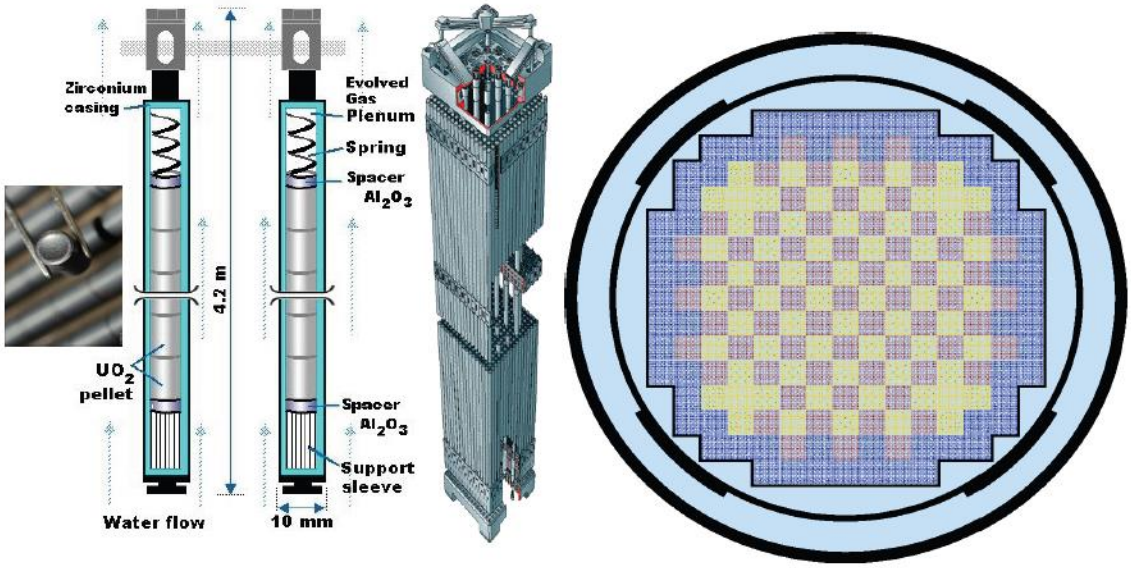


Figure 1.1 – LWR geometry

Additionally there is significant complexity in modeling some fuel rods. Some pellets, known as integrated fuel burnable absorbers (IFBA™) are coated with a very thin layer (< 10 microns) of ZrB_2, which is a strong neutron absorber and must be modeled explicitly.

Once the geometry is faithfully represented, the additional challenges of modeling a real reactor typically involve the inclusion of other physics which include: fuel depletion, thermal hydraulics and mechanics, heat transfer, and energy deposition from prompt fission gamma rays just to name a few. The focus of the work in this thesis is to advance the application of a 3-D transport method that is faithful to the physical geometry by making better use of modern computer architectures.

1.2 Summary of State of the Art 3-D Transport Methods

Several 3-D neutron transport methods have been developed over the years for performing the core calculation. This section provides a brief survey of the current state of the art methods as context for the work that is proposed for this thesis. The steady-state, 3-D Boltzmann neutron transport equation to describe the neutron flux in a reactor is given as:

$$\vec{\Omega}\cdot\nabla\varphi\left(\vec{r},\vec{\Omega},E\right)+\Sigma_t\left(\vec{r},E\right)\varphi\left(\vec{r},\vec{\Omega},E\right)= \frac{\chi(E)}{4\pi k_{eff}}\int_0^{\infty}\nu\Sigma_f\left(\vec{r},E'\right)\int_0^{4\pi}\varphi\left(\vec{r},\vec{\Omega}',E'\right)d\Omega'dE'+ \int_0^{\infty}\int_0^{4\pi}\Sigma_s\left(\vec{r},\vec{\Omega}'\cdot\vec{\Omega},E'\rightarrow E\right)\varphi\left(\vec{r},\vec{\Omega}',E'\right)d\Omega'dE', \quad \textbf{Eq. (1.1)}$$

where the standard notation, $\vec{r}$, $\vec{\Omega}$, E, is used for the space, angle, and energy variables, respectively. φ, Σ represent the angular neutron flux, and cross sections, respectively. For the cross sections the subscript indicates the reaction type, where *t* denotes total, *s* denotes scattering, and *f* denotes fission. χ is the normalized fission spectrum, and k_{eff} is the effective neutron multiplication factor, or eigenvalue, of the system.

Traditionally transport methods can be divided into two relatively broad categories: stochastic and deterministic; as illustrated in Figure 1.2.

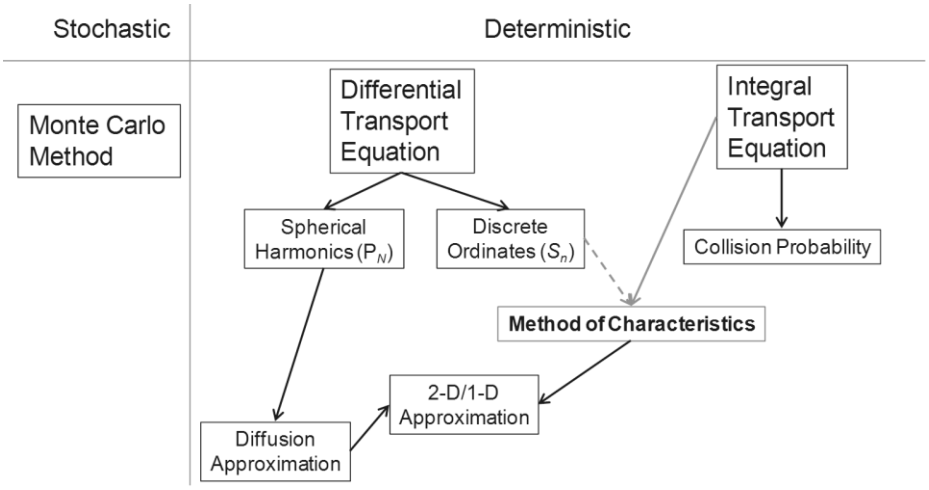


Figure 1.2 – Taxonomy of methods for solving the Boltzmann neutron transport equation

In the following sections, a brief overview is given for each method. This overview highlights the key approximations that are used in each of the methods and discusses their relative strengths and weaknesses. The chapter then ends with a summary of key points for why the thesis research on the 3-D Method of Characteristics (MOC) is a significant contribution to research on high fidelity transport calculations for full-core LWR applications.

1.2.1 Monte Carlo

The primary stochastic method is referred to as the *Monte Carlo* method. This method simulates individual particle transport by generating random numbers and sampling probability distributions for particle interactions that depend on the physics being modeled. Monte Carlo is one of the simplest and most reliable methods of describing neutron transport, since it can simulate the complicated physics of neutron and material interactions in arbitrary geometries with relatively little approximation. Another major advantage of this method is the detailed treatment of the energy variable through the random sampling of the continuous energy cross section data. However, there are several reasons why the method has been limited in its application to practical LWR reactor analysis.

The first principal reason is that a stochastic process always includes a statistical error in whatever quantity is being calculated. Most often the reactor analyst is interested in the power distribution inside a reactor. Depending on the desired spatial fidelity of this field quantity, the number of particles that must be tallied in each spatial mesh region to sufficiently reduce the statistical error in the tally may require the simulation of an enormous number of particles, which consequently increases the computational time [8]. A second related issue is the convergence of Monte Carlo methods for practical reactor problems. Unless some special steps are taken, the fission source may require an impractical number of histories to converge for some large problems, due to statistical noise and a high dominance ratio [12]. A third issue for using Monte Carlo methods for practical LWR simulation methods is the difficulty in modeling detailed thermal-fluid feedback for temperature and density. The feedback is a function of the neutron and power distribution, and since the neutron cross sections are a function of the temperature and density, this results in a set of non-linear coupled equations. Typically, some form of approximate method is necessary to treat thermal-fluid feedback with Monte Carlo methods. However, recent research has been successful in explicitly modeling the temperature effects on the cross section within the Monte Carlo code. Such “on-the-fly” temperature feedback may provide a breakthrough to help overcome this issue for power reactor simulation. Finally, should the aforementioned issues all be resolved for the steady-state reactor problem, they must then be extended to the time-dependent reactor in

order to be useful for practical LWR safety analysis, which is yet another significant challenge.

During the past several years, Monte Carlo researchers have developed several innovative solutions addressing several of the issues identified [9], [12], [13]. In fact, many now consider Monte Carlo methods to be a viable method for practical reactor analysis, and full core calculations have been demonstrated with several existing codes [14]. However, several issues must be resolved before Monte Carlo becomes widely accepted for practical LWR analysis. Consequently, continued investigation into deterministic methods for full core 3-D transport is well warranted and remains an important part of the research portfolio for CASL.

1.2.2 Discrete Ordinates

One of the oldest deterministic transport methods that was developed was the *discrete ordinates* or S_n method [15]. All deterministic methods are based on the discretization of each of the independent solution variables and the discrete ordinates refers to the discretization of the angular dependence of the neutron flux. The starting point in the derivation of the S_n method is to discretize the neutron energy into G energy groups by applying the multi-group approximation to Eq. (1.1). This results in the multi-group transport equation for a group, g, which can then be written as:

$$\vec{\Omega}\cdot\nabla\varphi_g\left(\vec{r},\vec{\Omega}\right)+\Sigma_{t,g}\left(\vec{r}\right)\varphi_g\left(\vec{r},\vec{\Omega}\right)=$$
$$\frac{\chi_g}{4\pi k_{eff}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}\left(\vec{r}\right)\int_0^{4\pi}\varphi_{g'}\left(\vec{r},\vec{\Omega}'\right)d\Omega'+\sum_{g'=1}^{G}\int_0^{4\pi}\Sigma_{s,g'\to g}\left(\vec{r},\vec{\Omega}'\cdot\vec{\Omega}\right)\varphi_{g'}\left(\vec{r},\vec{\Omega}'\right)d\Omega'. \qquad \textbf{Eq. (1.2)}$$

It should also be noted that the multi-group approximation may be used by the Monte Carlo method, but it is not preferred over continuous energy treatments. However, all of the deterministic methods described in this section begin with the multi-group approximation shown in Eq. (1.2), since continuous energy treatment is not practical.

The main approximation that leads to a discrete ordinates method is to choose a set of discrete angles or directions of flight. The accuracy of this approximation is determined largely by the quadrature used to compute the integrals over $\vec{\Omega}$. The "n" in S_n indicates the order of the polynomial or function that may be integrated exactly with a

given quadrature. The quadrature approximation is shown in Eq. (1.3), and is applied to the angular variable in Eq. (1.2). This leads to the discrete ordinates form of the multi-group transport equation given in Eq. (1.4).

$$\int_{4\pi} f(\vec{\Omega})d\vec{\Omega} \approx \sum_{m=1}^{M} w_m f(\vec{\Omega}_m), \qquad \textbf{Eq. (1.3)}$$

$$\left(\Omega_{x,m}\frac{\partial}{\partial x}+\Omega_{y,m}\frac{\partial}{\partial y}+\Omega_{z,m}\frac{\partial}{\partial z}\right)\varphi_{g,m}(\vec{r})+\Sigma_{t,g}(\vec{r})\varphi_{g,m}(\vec{r})= \frac{\chi_g}{4\pi k_{eff}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}(\vec{r})\sum_{m'=1}^{M}w_{m'}\varphi_{g',m'}(\vec{r})+\sum_{g'=1}^{G}\sum_{m'=1}^{M}w_{m'}\Sigma_{s,g'\to g,m'\to m}(\vec{r})\varphi_{g',m'}(\vec{r}), \qquad \textbf{Eq. (1.4)}$$

where *m* indicates the discrete ordinate.

For the last fifty years, the S_n method has been continuously developed with a variety of spatial discretization techniques, that include the full range of finite difference and finite element approximations. However, the finite difference approximation appears to be more commonly used; and when making use of this approximation an additional set of "auxiliary" equations must be developed to relate surface averaged angular flux quantities to volume averaged angular flux quantities. In general, the finite difference method is limited in its ability to model arbitrary geometries. Either a different discretized form of the equation must be derived using a specific approximation of the spatial derivative or approximations to the physical geometry must be made. Two of the discrete ordinates codes which have been adapted to massively parallel computing and been applied successfully to full core reactor analysis are the ORNL and ANL codes, Denovo [6] and UNÌC [7], respectively.

Denovo employs a first order finite difference approximation with a native mesh representation in a structured Cartesian grid. This imposes some limitations for reactor geometries that include cylindrical fuel pins. For performing its transport sweep, Denovo employs the well-established parallel KBA wave front algorithm [16] for spatial decomposition, which has shown some limitations in scalability for massive numbers of processors. A technique for parallelism in energy has also been recently implemented into the code which has been shown to scale quite well on the Cray XT5 machine Jaguar

[17]. Because of its scalability using space-angle-energy decomposition, the Denovo code has had several successes with large scale applications. However, Denovo has required prohibitive computational resources in order to resolve circular fuel pin boundaries, which are important for several of the relevant LWR problems. Therefore issues remain about the viability of Denovo's implementation of the Cartesian geometry based discrete ordinates method for widespread practical reactor analysis.

UNÌC on the other hand uses a finite element approximation and solves the even-parity form of the transport equation, which is second order in space; with a native mesh representation composed of arbitrary structured tetrahedrons and hexahedrons. This mesh representation is more suitable for representing the physical problem geometry. The solution algorithm in UNÌC poses the problem as a 1-group fixed source transport problem and represents this linear system with a large sparse matrix for each angle. This system is solved using the Conjugate Gradient method with preconditioning from the PETSc software library [18]. Because the PETSc solvers have been well designed and developed to perform on leadership class computers, the scaling of UNÌC on the IBM Blue Gene/P machines Intrepid [19], and JUGENE [20] and Cray XT5 machine Jaguar [17] is quite excellent, achieving >90% scalability up to ~130,000 cores. It is also worth noting that the S_n solver in UNÌC was a finalist for the Gordon Bell Prize in 2009 [21]. Presently, only space and angle are decomposed, and UNÌC does not employ energy decomposition. Because of this and other issues, considerable work remains in demonstrating the viability of UNÌC for practical full core LWR applications.

1.2.3 Spherical Harmonics

Spherical harmonics, or P_N, methods differ from S_n methods by making a different angular approximation. Instead of considering discrete directions of flight, the terms of Eq. (1.2) that are functions of $\vec{\Omega}$ are expanded into moments of the spherical harmonic functions. This is shown below for the multi-group angular flux:

$$\varphi_g\left(\vec{r},\vec{\Omega}\right)=\sum_{n=0}^{\infty}\frac{2n+1}{4\pi}\sum_{m=-n}^{n}Y_n^m\left(\vec{\Omega}\right)\varphi_{g,n,m}\left(\vec{r}\right). \qquad \textbf{Eq. (1.5)}$$

With the spherical harmonics expansion, the series must be truncated, which introduces the P_N approximation. The same spatial discretization techniques used in the S_n method are also applicable to the P_N method. In practical implementations the order of P_N may be varied and for extremely accurate results something like P_{23} may be needed for reactor problems [22]. The P_N method has never been widely used in the reactor physics community, primarily because the form of the P_N equations is more complicated compared to other methods. It also cannot treat material discontinuities or void regions well in certain situations, both of which are required for reactor analysis. Furthermore, the higher angular moments of the flux do not have a straightforward physical interpretation.

Nonetheless the UNÌC code [7] has an implementation of this method for 3-D whole core analysis. The implementation of the P_N solver in UNÌC is similar to the S_n solver, where a sparse linear system is built and solved iteratively with an efficient implementation of a conjugate gradient method [18]. Unfortunately, the performance of the P_N solver has not been studied systematically in the same way as the S_n solver, nor has its computational performance been analyzed in detail. This makes it somewhat harder to quantitatively assess the viability of the P_N transport method for practical LWR analysis. However, because the P_N equations have issues treating material discontinuities and voids, and do not always have a straightforward physical interpretation, it is likely that this method will remain to have limited use.

1.2.4 Collision Probability Method

The *collision probability*, CP, method [23] differs fundamentally from the deterministic methods described thus far, since it is based on discretizing an *integral* form, rather than a *differential* form, of the Boltzmann transport equation. To obtain the integral form of the transport equation, Eq. (1.2) is integrated over all angle and space. Additionally, from this integration, only the isotropic scattering can be represented explicitly thus leading to:

$$\phi_g(\vec{r}) = \int_R \frac{e^{\Sigma_{t,g}(\vec{r}')|\vec{r}-\vec{r}'|}}{4\pi|\vec{r}-\vec{r}'|}\left[\sum_{g'=1}^{G}\frac{\chi_g}{k_{eff}}\nu\Sigma_{f,g'}(\vec{r}') + \Sigma_{s,0,g'\to g}(\vec{r}')\right]\phi_g(\vec{r}')d\vec{r}'. \qquad \textbf{Eq. (1.6)}$$

The collision probability method then recasts Eq. (1.6) into terms of the collision probabilities P_{ij} between the two discrete volumes V_i and V_j. These probabilities must then be determined numerically prior to the iterative solution scheme.

$$\phi_{g,i} = \sum_{j=1}^{J} P_{g,i,j}\frac{\left[\sum_{g'=1}^{G}\frac{\chi_g}{k_{eff}}\nu\Sigma_{f,g',j} + \Sigma_{s,0,g'\to g,j}\right]\phi_{g,j}}{\Sigma_{t,g,i}V_i\Sigma_{t,g,j}}. \qquad \textbf{Eq. (1.7)}$$

The method has been used successfully in analyzing 2-D assembly-sized problems [24], [25], [26] to generate few group cross sections for lower order full core calculations. The advantages of this method are that it gives very accurate results and can easily treat completely arbitrary geometries. The eventual drawback of the method, which has seen a decline in popularity in the reactor physics community, is that the coefficient matrix for the transmission probabilities is fully dense and is the size of the square of the number of discrete spatial regions. Although the theory for extending this method to 3-D is very straightforward, there has not been much research into the method for 3-D applications, most likely because of the inherent computational costs. However, interface current techniques have been developed that allow one to decouple these collision probability matrices for different subdomains [24], [28]. Although, the interface current techniques require some approximations be introduced for the angular order of the currents, the CP method still has potential for 3-D whole core analysis but would require innovative methods for accurately treating interface currents. CP also suffers from limitations in representing the scattering source, since the integral equation is limited to isotropic scattering. Research would be required to develop innovative methods to treat anisotropic scattering within the framework of CP methods, since anisotropic scattering can be important for practical LWR applications.

1.2.5 Method of Characteristics

The *method of characteristics* [29], MOC, is perhaps the newest of the transport solution techniques discussed so far. It was not first used in a production tool until 1980 [30]. Over the last several years it has been implemented in most of the popular lattice physics codes [2], [3], [27] and it has become the most popular transport method for routine 2-D assembly level analysis to generate cross sections for practical full core LWR simulations.

The MOC solution is a general mathematical technique for solving partial differential equations. For the transport equation, a coordinate transformation is applied to Eq. (1.2) to yield a first order ordinary differential equation for the solution along the "characteristic direction".

$$\frac{d\varphi_g}{ds}\left(\vec{r}_0 + s\vec{\Omega}, \vec{\Omega}\right) + \Sigma_{t,g}\left(\vec{r}_0 + s\vec{\Omega}\right)\varphi_g\left(\vec{r}_0 + s\vec{\Omega}, \vec{\Omega}\right) = q\left(\vec{r}_0 + s\vec{\Omega}, \vec{\Omega}, E\right), \qquad \textbf{Eq. (1.8)}$$

where,

$$\begin{aligned} q\left(\vec{r}_0 + s\vec{\Omega}, \vec{\Omega}, E\right) = {} & \frac{\chi(E)}{4\pi k_{eff}} \int_0^{\infty}\int_0^{4\pi} \nu\Sigma_f\left(\vec{r}_0 + s\vec{\Omega}', E'\right)\varphi\left(\vec{r}_0 + s\vec{\Omega}', \vec{\Omega}', E'\right)d\Omega' dE' \\ & + \int_0^{\infty}\int_0^{4\pi} \Sigma_s\left(\vec{r}_0 + s\vec{\Omega}', \vec{\Omega}'\cdot\vec{\Omega}, E' \rightarrow E\right)\varphi\left(\vec{r}_0 + s\vec{\Omega}', \vec{\Omega}', E'\right)d\Omega' dE'. \end{aligned} \qquad \textbf{Eq. (1.9)}$$

This equation can then be integrated analytically along the characteristic direction for a homogenous region, provided some assumption is made regarding the shape of the source within the region. This leads to an equation for the propagation or transmission of the angular flux through a domain, which can then be integrated over some length to provide an expression for the average angular flux along the characteristic in the given region. These equations are shown below:

$$\varphi_{i,g,m,k}^{out} = \varphi_{i,g,m,k}^{in}\exp\left(-\Sigma_{t,i,g}s_{i,k,m}\right) + \frac{q_{i,g}}{\Sigma_{t,i,g}}\left[1 - \exp\left(-\Sigma_{t,i,g}s_{i,k,m}\right)\right], \qquad \textbf{Eq. (1.10)}$$

$$\overline{\varphi}_{i,g,m,k} = \frac{\varphi^{in}_{i,g,m,k} - \varphi^{out}_{i,g,m,k}}{\Sigma_{t,i,g} s_{i,m,k}} + \frac{q_{i,g}}{\Sigma_{t,i,g}}. \qquad \textbf{Eq. (1.11)}$$

Consequently, the problem must be discretized in the characteristic space, which means discrete directions of flight must be chosen. For a given direction, several characteristic lines or rays must be tracked through a discretized spatial domain. One notable difference of this method, compared to the CP method, is that in most implementations a linear system is never formed and instead a *transport sweep* is performed. An advantage of MOC is that it also readily handles any arbitrary geometry, provided one knows how to intersect a series of lines with surfaces of the geometry. The MOC method is typically thought to be superior to the CP method because it does not have the same restriction of assuming isotropic scattering. In fact extending the MOC source to higher orders in space or angle is relatively straightforward, which is also an advantage of the S_n method. Finally, the MOC method enjoys the same advantages of deterministic methods compared to Monte Carlo methods.

While MOC is the preferred 2-D method for reactor analysis and considerable research has been devoted to developing efficient 2-D kernels, the 3-D MOC method is generally viewed as the most computationally expensive of the 3-D transport methods. This is because when adding the third dimension, the number of rays that must be tracked and the number of discrete spatial regions required to accurately model a problem increase the computational requirements of the 2-D problem by a factor on the order of 1000. Nonetheless there has been notable research into the 3-D MOC method [7], [31], [32], [33] with some efforts focusing on parallelism [7], [34].

The non-parallel research [31], [32], [33] helped to demonstrate the efficacy of the modular ray tracing concept. The definition and implementation of this, used in the research here, is discussed in detail in Chapter 2. In the parallel algorithm research done in the DRAGON [34] code, the approach taken was to focus only on parallelism by angle and by ray. This work showed modest parallelism up to $O(100)$ processors, which were reasonably large machines for their time. In the UNÌC [7] code a similar approach was taken at first, but eventually abandoned because of its inability to scale on petaflop machines. The new approach used in the UNÌC code was to make use of parallelism in

space and angle by forming a blocked linear system and then solving this using a hierarchy of GMRES solvers in PETSc. However, unlike the other transport methods implemented in UNÌC, the parallel performance of the MOC solver was observed to be relatively poor due primarily to issues of load imbalance, and further research was suggested.

1.3 State of the Art Approximate 3-D Transport Methods

In addition to 3-D transport methods, there are three other classes of methods that are relevant to the discussion of full core LWR analysis. The first class of approximate methods are based on the diffusion approximation which has been the work horse for full core analysis for nearly a half-century. The next is the simplified P_N method which is similar to the P_N method described previously, and the last class of methods are commonly referred to as the 2-D/1-D method, fusion method, or sometimes synthesis method. In this document the method is referred to as 2-D/1-D. This section discusses the key approximations made that characterize each of these methods and their relative strengths and weaknesses compared to 3-D transport.

1.3.1 Diffusion

In diffusion based methods the key approximation is to eliminate the angular dependence of the neutron flux. The diffusion equation, given in Eq. (1.12), can be derived from the multi-group transport equation, Eq. (1.2), by first computing the zero-th and first angular moments of Eq. (1.2). Next approximations are introduced to the second order terms to provide closure. This yields the P_1 equations. For diffusion methods an additional approximation is introduced. In the diffusion approximation, the anisotropic scattering is assumed to be only within-group and not group to group. However, it should be noted that for some problems the diffusion equation is mathematically consistent with the P_1 approximation of Eq. (1.2).

$$-\nabla\cdot\left(D_g(\vec{r})\nabla\phi_g(\vec{r})\right)+\Sigma_{t,g}(\vec{r})\phi_g(\vec{r})= \left[\sum_{g'=1}^{G}\left(\Sigma_{s0,g'\to g}(\vec{r})+\frac{\chi_g}{k_{eff}}\nu\Sigma_{f,g'}(\vec{r})\right)\phi_{g'}(\vec{r})\right]. \quad \textbf{Eq. (1.12)}$$

From Eq. (1.12), one may note that the primary approximation introduced affects the leakage term. Another heuristic derivation of the diffusion equation involves introducing *Fick's Law*:

$$\vec{J}_g(\vec{r})=D_g(\vec{r})\nabla\phi_g(\vec{r}). \quad \textbf{Eq. (1.13)}$$

Consequently, because the diffusion approximation simplifies the leakage term it break downs in regions or problems with strong gradients in the flux, but otherwise the method is extremely computationally efficient compared to transport because it does treat the angular dependence of the neutron flux explicitly. The diffusion method can be viewed as the lowest order approximation that is acceptable for solving the Boltzmann equation and is commonly viewed as the classical method for reactor analysis. It is relevant to the larger discussion of whole core LWR analysis because any 3-D transport method should be at least as accurate as 3-D diffusion, so in that sense it provides the upper bound for inaccuracy in the solution. Finally, a large body of work exists which shows how the diffusion equation can be used as an efficient low order operator to accelerate the convergence of a transport method, and since 3-D transport methods are the most computationally expensive, this form of acceleration becomes more important.

1.3.2 Simplified P_N

The Simplified P_N (SP_N) approximation [35] is a leading order asymptotic limit of the transport equation. One way in which the SP_N equations can derived from the P_N equations is by starting with the 1-D P_N equations and replacing the 1-D diffusion operator with the 3-D diffusion operator. This is equivalent to replacing the first order derivatives of the odd moments of the angular flux with divergence operators, and the first order derivatives of the even moments with gradient operators. The SP_3 equations are given below:

$$-\nabla\cdot D_{0,g}(\vec{r})\nabla\Phi_{0,g}(\vec{r})+\left[\Sigma_t(\vec{r})-\Sigma_{s0}(\vec{r})\right]\Phi_{0,g}(\vec{r})= 2\left[\Sigma_t(\vec{r})-\Sigma_{s0}(\vec{r})\right]\Phi_{2,g}(\vec{r})+q_g(\vec{r}), \quad \textbf{Eq. (1.14)}$$

$$-\nabla\cdot D_{2,g}(\vec{r})\nabla\Phi_{2,g}(\vec{r})+\left[\Sigma_t(\vec{r})-\Sigma_{s2}(\vec{r})\right]\Phi_{2,g}(\vec{r})= \frac{2}{5}\left[\Sigma_t(\vec{r})-\Sigma_{s0}(\vec{r})\right]\left[\Phi_{0,g}(\vec{r})-2\Phi_{2,g}(\vec{r})\right]+q_g(\vec{r}), \quad \textbf{Eq. (1.15)}$$

where the diffusion coefficients D_0 and D_2 are those derived from the 1-D P_3 equations and the scalar flux is computed from the 0th and 2nd order angular moments of the flux:

$$\phi_g(\vec{r})=\Phi_{0,g}(\vec{r})-2\Phi_{2,g}(\vec{r}). \quad \textbf{Eq. (1.16)}$$

The main advantage of the SP_N method is that the structure and implementation of the equations is very similar to the diffusion equation, thus if one has an existing diffusion solver, it can be easily modified to be able to solve the SP_N equations. Furthermore, the SP_N approximation can be considerably more accurate than the diffusion approximation for a wider range of problems, and has been implemented successfully in some core simulators [36], [37].

1.3.3 2-D/1-D

Perhaps the most recent advance in approximations of 3-D transport is the 2-D/1-D method [38], [39], [40]. The principal motivation for this method is that generally there is more heterogeneity in the LWR geometry in the radial (2-D) plane than in the axial (1-D) direction. This suggests that a lower order method could be used to solve the axial problem compared to the radial problem, and a "transverse leakage" could be used to couple the two solutions.

This method starts with the multi-group transport equation and introduces an approximation to the derivative in the *z*-direction.

$$\Omega_x \frac{\partial}{\partial x}\varphi_g\left(\vec{r},\vec{\Omega}\right)+\Omega_y \frac{\partial}{\partial y}\varphi_g\left(\vec{r},\vec{\Omega}\right)+\frac{\partial}{\partial z}F_g\left(\vec{r},\vec{\Omega}\right)+\Sigma_{t,g}\left(\vec{r}\right)\varphi_g\left(\vec{r},\vec{\Omega}\right)=$$
$$\frac{\chi_g}{4\pi k_{eff}}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}\left(\vec{r}\right)\int_0^{4\pi}\varphi_{g'}\left(\vec{r},\vec{\Omega}'\right)d\Omega'+\sum_{g'=1}^{G}\int_0^{4\pi}\Sigma_{s,g'\to g}\left(\vec{r},\vec{\Omega}'\cdot\vec{\Omega}\right)\varphi_{g'}\left(\vec{r},\vec{\Omega}'\right)d\Omega', \quad \textbf{Eq. (1.17)}$$

where $F_g\left(\vec{r},\vec{\Omega}\right)$ is some reasonable approximation to the axial derivative such as:

$$\frac{\partial}{\partial z}F_g\left(\vec{r},\vec{\Omega}\right)=-\frac{\partial}{\partial z}\frac{D_g}{4\pi}\frac{\partial}{\partial z}\phi_g\left(\vec{r}\right)\approx\Omega_z\frac{\partial}{\partial z}\varphi_g\left(\vec{r},\vec{\Omega}\right). \quad \textbf{Eq. (1.18)}$$

In previous work the function *F* has been treated with a variety of approximations including the an S_n approximation, the SP_N approximation and the diffusion approximation. Even with the lowest order approximation (diffusion), shown in Eq. (1.18), the 2-D/1-D method has been shown to preserve the 2-D transport equation and the 1-D (in *z*) and 3-D diffusion equation [40]. The advantage of the 2-D/1-D method is that it assumes the solution is weakly varying in the *z*-direction allowing for coarser discretizations in *z* and only having to discretize several 2-D domains. Again, for most LWR reactor designs the heterogeneity of the geometry is largely in the *x*-*y* plane and the geometry is fairly uniform in *z*, thus making the approximation of the 2-D/1-D method likely a good approximation for these problems. However, the accuracy of the method will suffer when transport effects are observed in the *z*-direction, and usually this corresponds to partially inserted control rods.

1.4 Summary

Of the methods surveyed in this section the Monte Carlo, the S_n, and the MOC methods have received the most attention throughout the research community for consideration in 3-D full core LWR analysis. Parallel algorithms for the S_n method appear to be the most mature in terms of being able to scale well on petascale machines. However, the Monte Carlo method is not far behind S_n in its ability to map onto massively parallel architectures [41]. Both of these methods are still under active research in order to address the remaining issues noted previously. However, it is questionable whether an acceptable solution will present itself for the ability of the S_n method to deal with

complex geometries (e.g. curvilinear surfaces) and maintain the current parallel performance. The issues remaining for the Monte Carlo method are more about improving techniques for capturing the physics of a reactor accurately, namely the efficient computation of state vectors for high fidelity spatial discretizations with minimal statistical error, modeling of thermal-fluid feedback, and the treatment of time dependence. Conventional wisdom suggests that pursuing development of P_N and CP methods for full core 3-D LWR analysis may not be worthwhile because of the inherent drawbacks of the methods themselves.

This leaves 3-D MOC as an area of research that may still have potential for practical LWR applications. As described in the previous section, past efforts have not been very successful in developing highly scalable parallel 3-D MOC algorithms. Therefore, the principal challenge and focus of this thesis research is the development of a highly scalable parallel algorithm for solving the 3-D neutron transport equation by the method of characteristics for practical LWR applications.

The rest of this thesis proceeds by first presenting the detailed derivation of the MOC solution of the 3-D Boltzmann transport equation in Chapter 2, along with the details for the discretization and iteration scheme to obtain a numerical solution. Chapter 3 describes the actual implementation of the 3-D MOC kernel used in this work, along with some important optimizations. Chapter 4 presents the details of how the kernel described in Chapter 3 is parallelized. Chapter 5 introduces a performance model to analyze and predict the performance of the 3-D MOC solution algorithm. The validation of this model is also presented in Chapter 5. The focus of Chapter 6 is the detailed analysis of the parallel 3-D MOC kernel's performance on a few architectures. Chapter 7 then discusses how the convergence of the algorithm may be improved by extending well known diffusion-based acceleration techniques. Chapter 8 presents the results and performance of the parallel 3-D MOC algorithm for a select number of numerical benchmark problems with comparisons to 2-D/1-D results. Finally, the conclusions of this work are given in Chapter 9.

Chapter 2

FUNDAMENTALS OF THE METHOD OF CHARACTERISTICS

This chapter provides a detailed derivation of the discretized MOC equations and introduces several of the concepts and algorithms conventionally used in MOC solvers. First, a detailed derivation is provided, which highlights important approximations at each step. Next, the algorithm for the iterative solution of these equations is described. Then, the techniques required to discretize a problem that are common to any multi-dimensional MOC transport solver are described. The descriptions primarily focus on 2-D solvers at first, and then the concepts are extended to the case of a 3-D solver to highlight the additional challenges that must be addressed. Finally, detailed examples, again given in the context of 2-D, in which the MOC transport sweep can be performed, are presented and contrasted to highlight their relative merits. This establishes the basis for a 3-D MOC serial algorithm that can then be parallelized.

2.1 Method of Characteristics Solution of the Boltzmann Transport Equation in 3-D

The derivation of the MOC solution to the Boltzmann neutron transport equation starts with the steady-state continuous form of the equation given in Eq. (2.1) below.

$$\vec{\Omega}\cdot\nabla\varphi\left(\vec{r},\vec{\Omega},E\right)+\Sigma_t\left(\vec{r},E\right)\varphi\left(\vec{r},\vec{\Omega},E\right)= \frac{\chi(E)}{4\pi k_{eff}}\int_0^\infty \nu\Sigma_f\left(\vec{r},E'\right)\int_0^{4\pi}\varphi\left(\vec{r},\vec{\Omega}',E'\right)d\Omega'dE' + \int_0^\infty\int_0^{4\pi}\Sigma_s\left(\vec{r},\vec{\Omega}'\cdot\vec{\Omega},E'\to E\right)\varphi\left(\vec{r},\vec{\Omega}',E'\right)d\Omega'dE'. \quad \textbf{Eq. (2.1)}$$

Eq. (2.1) is the same as Eq. (1.1) from the previous chapter. Next the variable q is introduced to simplify the right hand side.

$$q\left(\vec{r},\vec{\Omega},E\right)=\frac{\chi\left(\vec{r},E\right)}{4\pi k_{eff}}\int_0^\infty \nu\Sigma_f\left(\vec{r},E'\right)\int_0^{4\pi}\varphi\left(\vec{r},\vec{\Omega}',E'\right)d\Omega'dE' + \int_0^\infty\int_0^{4\pi}\Sigma_s\left(\vec{r},\vec{\Omega}'\cdot\vec{\Omega},E'\to E\right)\varphi\left(\vec{r},\vec{\Omega}',E'\right)d\Omega'dE', \quad \textbf{Eq. (2.2)}$$

yielding

$$\vec{\Omega}\cdot\nabla\varphi\left(\vec{r},\vec{\Omega},E\right)+\Sigma_t\left(\vec{r},E\right)\varphi\left(\vec{r},\vec{\Omega},E\right)=q\left(\vec{r},\vec{\Omega},E\right). \quad \textbf{Eq. (2.3)}$$

2.1.1 Transformation to the Characteristic Direction

Now, the method of characteristics is applied, in which the spatial and angle variables of the partial differential equation, Eq. (2.3), are transformed into the *characteristic direction* using the following identities.

$$\vec{r}=\vec{r}_0+s\vec{\Omega}\Rightarrow\begin{aligned}x(s)&=x_0+s\Omega_x\\ y(s)&=y_0+s\Omega_Y\\ z(s)&=z_0+s\Omega_z\end{aligned}. \quad \textbf{Eq. (2.4)}$$

This leads to the characteristic form of Eq. (2.3) shown in Eq. (2.5).

$$\frac{d\varphi}{ds}\left(\vec{r}_0+s\vec{\Omega},\vec{\Omega},E\right)+\Sigma_t\left(\vec{r}_0+s\vec{\Omega},E\right)\varphi\left(\vec{r}_0+s\vec{\Omega},\vec{\Omega},E\right)=q\left(\vec{r}_0+s\vec{\Omega},\vec{\Omega},E\right), \quad \textbf{Eq. (2.5)}$$

where

$$q\left(\vec{r}_0 + s\vec{\Omega}, \vec{\Omega}, E\right) = \frac{\chi\left(\vec{r}_0 + s\vec{\Omega}, E\right)}{4\pi k_{eff}} \int_0^{\infty} \int_0^{4\pi} \nu\Sigma_f\left(\vec{r}_0 + s\vec{\Omega}', E'\right)\varphi\left(\vec{r}_0 + s\vec{\Omega}', \vec{\Omega}', E'\right) d\Omega' dE'$$
$$+ \int_0^{\infty} \int_0^{4\pi} \Sigma_s\left(\vec{r}_0 + s\vec{\Omega}', \vec{\Omega}' \cdot \vec{\Omega}, E' \to E\right)\varphi\left(\vec{r}_0 + s\vec{\Omega}', \vec{\Omega}', E'\right) d\Omega' dE'. \quad \textbf{Eq. (2.6)}$$

Eq. (2.5) can then be solved analytically using the following integrating factor:

$$\exp\left(-\int_0^s \Sigma_t\left(\vec{r}_0 + s'\vec{\Omega}, E\right) ds'\right),$$

which leads to:

$$\varphi\left(\vec{r}_0 + s\vec{\Omega}, \vec{\Omega}, E\right) = \varphi\left(\vec{r}_0, \vec{\Omega}, E\right)\exp\left(-\int_0^s \Sigma_t\left(\vec{r}_0 + s'\vec{\Omega}, E\right) ds'\right)$$
$$+ \int_0^s q\left(\vec{r}_0 + s'\vec{\Omega}, \vec{\Omega}, E\right)\exp\left(-\int_{s'}^s \Sigma_t\left(\vec{r}_0 + s''\vec{\Omega}, E\right) ds''\right) ds'. \quad \textbf{Eq. (2.7)}$$

Eq. (2.7) is the solution of the characteristics form of the continuous Boltzmann transport equation. Next, this equation is discretized so that it may be solved numerically. Some reasonable approximations are introduced to accomplish this, and also so that the integrals of Eq. (2.7) may be evaluated more easily.

2.1.2 The Multi-group Approximation

The first approximation to be introduced, which is common for almost all deterministic methods, is the *multi-group approximation*. This discretizes the energy variable by defining discrete neutron energy groups. The multi-group cross sections are determined exactly by Eq. (2.8). However, $\varphi\left(\vec{r}, \vec{\Omega}, E\right)$ is generally not known *a priori*, therefore the approximation of Eq. (2.9) is introduced. The multi-group cross sections are then defined as shown in Eq. (2.10) using a weighting factor $\Psi\left(\vec{r}, E\right)$ in energy. This weighting factor should typically be representative of the neutron energy spectrum of the system, which cannot be known exactly for all potential problems *a priori*. As long as the energy distribution of the neutron flux in the system to be simulated is reasonably consistent with the weighting spectrum used to collapse the continuous energy cross section, the multi-

group approximation is accurate since it preserves the reaction rates within each energy group.

$$\Sigma_{x,g}\left(\vec{r}\right)=\frac{\int_{E_g}^{E_{g-1}}\Sigma_x\left(\vec{r},E\right)\varphi\left(\vec{r},\vec{\Omega},E\right)dE}{\int_{E_g}^{E_{g-1}}\varphi\left(\vec{r},\vec{\Omega},E\right)dE}, \qquad \textbf{Eq. (2.8)}$$

$$\varphi\left(\vec{r},\vec{\Omega},E\right)\approx\Psi\left(\vec{r},E\right)f\left(\vec{r},\vec{\Omega}\right), \qquad \textbf{Eq. (2.9)}$$

$$\Sigma_{x,g}\left(\vec{r}\right)\approx\frac{\int_{E_g}^{E_{g-1}}\Sigma_x\left(\vec{r},E\right)\Psi\left(\vec{r},E\right)dE}{\int_{E_g}^{E_{g-1}}\Psi\left(\vec{r},E\right)dE}, \qquad \textbf{Eq. (2.10)}$$

$$\chi_g\left(\vec{r}\right)=\int_{E_g}^{E_{g-1}}\chi\left(\vec{r},E\right)dE. \qquad \textbf{Eq. (2.11)}$$

Eq. (2.11) does not use the weighting factor since the fission spectrum is not strictly a cross section. In Eq. (2.10), the subscript *x* is to indicate a reaction type. Applying the multi-group approximation to Eq. (2.7) leads to the steady-state multi-group MOC solution of Boltzmann neutron transport equation shown in Eq. (2.10), where the subscript *g*, is introduced to denote the neutron energy group index.

$$\begin{aligned}\varphi_g\left(\vec{r}_0+s\vec{\Omega},\vec{\Omega}\right)=\varphi_g\left(\vec{r}_0,\vec{\Omega}\right)&\exp\left(-\int_0^s\Sigma_{t,g}\left(\vec{r}_0+s'\vec{\Omega}\right)ds'\right)\\&+\int_0^s q_g\left(\vec{r}_0+s'\vec{\Omega},\vec{\Omega}\right)\exp\left(-\int_{s'}^s\Sigma_{t,g}\left(\vec{r}_0+s''\vec{\Omega}\right)ds''\right)ds'.\end{aligned} \qquad \textbf{Eq. (2.12)}$$

2.1.3 The Discrete Ordinates Approximation

The next approximation that is introduced after the multi-group approximation is the *discrete ordinates approximation* for the angular variable. This is essentially a quadrature approximation, which for a given function of angle is written as:

$$\int_{4\pi} f\left(\vec{\Omega}\right)d\vec{\Omega} \approx \sum_{m=1}^{M} w_m f\left(\vec{\Omega}_m\right). \qquad \textbf{Eq. (2.13)}$$

Applying this approximation to Eq. (2.12) and Eq. (2.6) leads to:

$$\begin{aligned}\varphi_{g,m}\left(\vec{r}_0 + s\vec{\Omega}_m\right) = \varphi_{g,m}\left(\vec{r}_0\right)\exp\left(-\int_0^s \Sigma_{t,g}\left(\vec{r}_0 + s'\vec{\Omega}_m\right)ds'\right) \\ + \int_0^s q_{g,m}\left(\vec{r}_0 + s'\vec{\Omega}_m\right)\exp\left(-\int_{s'}^s \Sigma_{t,g}\left(\vec{r}_0 + s''\vec{\Omega}_m\right)ds''\right)ds',\end{aligned} \qquad \textbf{Eq. (2.14)}$$

where

$$\begin{aligned}q_{g,m}\left(\vec{r}_0 + s\vec{\Omega}_m\right) = \frac{\chi_g\left(\vec{r}_0 + s\vec{\Omega}_m\right)}{4\pi k_{eff}}\sum_{g'=1}^{G}\sum_{m'=1}^{M} \nu\Sigma_{f,g'}\left(\vec{r}_0 + s\vec{\Omega}_{m'}\right)w_{m'}\varphi_{g',m'}\left(\vec{r}_0 + s\vec{\Omega}_{m'}\right) \\ + \sum_{g'=1}^{G}\sum_{m'=1}^{M} w_{m'}\Sigma_{s,g'\to g}\left(\vec{r}_0 + s\vec{\Omega}_{m'}, \vec{\Omega}_{m'}\cdot\vec{\Omega}_m\right)\varphi_{g',m'}\left(\vec{r}_0 + s\vec{\Omega}_{m'}\right).\end{aligned} \qquad \textbf{Eq. (2.15)}$$

So long as the error introduced by Eq. (2.13) is minimal, the discrete ordinates approximation is valid. This approximation has been used for decades by the S_n and MOC methods, and in practice it is observed to be quite accurate if a sufficient number of angles are used.

2.1.4 Constant Material Properties in a Discrete Region

To discretize the spatial domain, the problem is divided into arbitrarily shaped discrete regions. Within each region it is assumed that the material properties are constant with respect to the spatial variable. This spatial discretization, as illustrated by Figure 2.1, essentially leads to a spatial discretization scheme that is first order accurate.

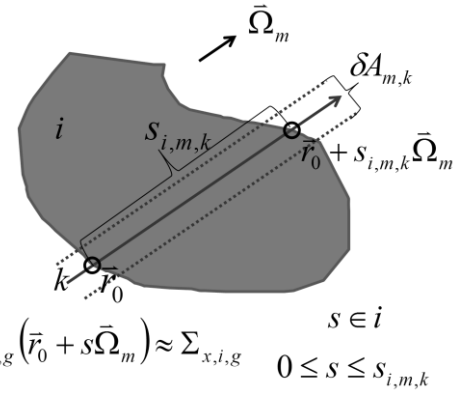


Figure 2.1 – Spatial discretization with constant properties

From these definitions and the constant material property, Eq. (2.14) and Eq. (2.15) are reduced to the following for each characteristic ray passing through each discrete region denoted by the subscripts *k* and *i*, respectively.

$$\varphi^{out}_{i,g,m,k} = \varphi^{in}_{i,g,m,k} \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right) + \int_0^{s_{i,m,k}} q_{i,g,m}(s') \exp\left(-\Sigma_{t,i,g}\left(s_{i,m,k} - s'\right)\right) ds', \quad \textbf{Eq. (2.16)}$$

$$q_{i,g,m}(s) = \frac{\chi_{i,g}}{4\pi k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_{f,i,g'} \sum_{m'=1}^{M} w_{m'} \varphi_{i,g',m'}(s) + \sum_{g'=1}^{G} \sum_{m'=1}^{M} w_{m'} \Sigma_{s,i,g' \to g}\left(\vec{\Omega}_{m'} \cdot \vec{\Omega}_m\right) \varphi_{i,g',m'}(s) \quad 0 \le s \le s_{i,m,k}. \quad \textbf{Eq. (2.17)}$$

In Eq. (2.16) the short hand notation, $\varphi^{in}_{i,g,m,k} = \varphi_{i,g,m}\left(\vec{r}_0\right) = \varphi_{i,g,m,k}\left(s = 0\right)$ and $\varphi^{out}_{i,g,m,k} = \varphi_{i,g,m}\left(\vec{r}_0 + s_{i,m,k}\vec{\Omega}_m\right) = \varphi_{i,g,m,k}\left(s = s_{i,m,k}\right)$, is used. For adjacent regions *i* and *i*+1 the identity, $\varphi^{out}_{i,g,m,k} = \varphi^{in}_{i+1,g,m,k}$, is also true.

2.1.5 Flat Source Region Approximation

Next the source, $q_{i,g,m}(s)$, is assumed to be constant within each discrete spatial region. This is commonly referred to as the *flat source* approximation. It is the lowest order approximation for the spatial dependence of the source and is accurate in the fine limit of the spatial mesh. Other approximations, such as linear and quadratic have been developed, but for the work here only the flat source is considered. With the flat source

approximation, the remaining integral over s' can be evaluated analytically, which leads to Eq. (2.18) and Eq. (2.19):

$$\varphi_{i,g,m,k}^{out} = \varphi_{i,g,m,k}^{in} \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right) + \frac{q_{i,g,m}}{\Sigma_{t,i,g}}\left[1-\exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right)\right], \qquad \textbf{Eq. (2.18)}$$

$$q_{i,g,m} = \frac{\chi_{i,g}}{4\pi k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_{f,i,g'} \sum_{m'=1}^{M} w_{m'} \overline{\varphi}_{i,g',m'} + \sum_{g'=1}^{G}\sum_{m'=1}^{M} w_{m'} \Sigma_{s,i,g'\to g}\left(\vec{\Omega}_{m'} \cdot \vec{\Omega}_{m}\right)\overline{\varphi}_{i,g',m'}. \qquad \textbf{Eq. (2.19)}$$

Eq. (2.19) introduces a new term, the region average angular flux, $\overline{\varphi}_{i,g,m}$. This is computed from the segment-average angular flux which is defined as:

$$\tilde{\varphi}_{i,g,m,k} = \frac{\int_0^{s_{i,m,k}} \varphi_{i,g,m}(s')ds'}{\int_0^{s_{i,m,k}} ds'} \Rightarrow \frac{\varphi_{i,g,m,k}^{in} - \varphi_{i,g,m,k}^{out}}{\Sigma_{t,i,g} s_{i,m,k}} + \frac{q_{i,g,m}}{\Sigma_{t,i,g}}. \qquad \textbf{Eq. (2.20)}$$

The region average angular flux is then computed from the segment-average angular fluxes as shown in Eq. (2.21), where $\delta A_{m,k}$ denotes the cross sectional area of the characteristic ray as illustrated in Figure 2.1.

$$\overline{\varphi}_{i,g,m} = \frac{\sum_{k\in i} \tilde{\varphi}_{i,g,m,k} s_{i,m,k} \delta A_{m,k}}{\sum_{k\in i} s_{i,m,k} \delta A_{m,k}}. \qquad \textbf{Eq. (2.21)}$$

Eq. (2.18) and Eq. (2.20) are the fundamental discretized MOC equations that must be evaluated to obtain a solution of the angular flux for a given source *q*.

2.1.6 Isotropic Scattering Approximation

The final approximation is to assume that the source is isotropic. Again this is the simplest approximation, and higher order angular sources have been derived, but for the work here only the isotropic source is considered. In general, the level of anisotropy of the source is represented by expanding the source as a function of Legendre polynomials. For an isotropic source, Eq. (2.19) reduces to the following form.

$$q_{i,g} = \frac{\chi_{i,g}}{4\pi k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_{f,i,g'}\phi_{i,g'} + \frac{1}{4\pi} \sum_{g'=1}^{G} \Sigma_{s0,i,g'\to g}\phi_{i,g'}, \qquad \textbf{Eq. (2.22)}$$

where the scalar flux, $\phi_{i,g}$ is computed as:

$$\phi_{i,g} = \int_{4\pi} \overline{\varphi}_{i,g}\left(\vec{\Omega}\right) \approx \sum_{m=1}^{M} w_m \overline{\varphi}_{i,g,m}. \qquad \textbf{Eq. (2.23)}$$

2.1.7 Iteration Scheme

In general the quantities of interest for reactor analysis such as reaction rates are determined from the scalar flux and not the angular flux, therefore the scalar flux is typically the primary solution variable updated by a transport kernel. Since any 3-D MOC kernel will make use of the above equations in some form, the kernel can be based on the concept of a 1-group fixed source problem which makes it possible to abstract the kernel into the following functional form:

$$\left(\vec{\varphi}_g^{in,(n+1)}, \vec{\phi}_g^{(n+1)}\right) = f\left(\vec{\varphi}_g^{in,(n)}, \vec{\bar{q}}_g^{(n)}\right), \qquad \textbf{Eq. (2.24)}$$

$$\overline{q}_{g,i}^{n+1} = \frac{1}{4\pi}\left(Q_{ext,i,g} + \Sigma_{s0,i,g\to g}\phi_{i,g}^{n+1}\right). \qquad \textbf{Eq. (2.25)}$$

In Eq. (2.24), $\vec{\varphi}_g^{in,(n)}$ is a vector of the discrete incoming angular flux boundary conditions in all space and angle for a single group, g. $\vec{\bar{q}}_g^{(n)}$ is a vector of the 1-group source computed as shown in Eq. (2.25) for all regions and $\vec{\phi}_g^{(n)}$ is a vector of the scalar fluxes for all regions. By alternately evaluating the function of Eq. (2.24) and updating the source defined by Eq. (2.25), it is possible to first define an *inner iteration* scheme to solve the 1-group fixed source problem for a given external source $Q_{ext,i,g}$. In these equations n is the inner iteration index and the algorithm for solving the 1-group fixed source problem is shown in Figure 2.2.

1. Guess initial source.
2. Compute outgoing angular fluxes by evaluating Eq. (2.18) for all segments
3. Compute segment-average angular flux by evaluating Eq. (2.20) for all segments.
4. Compute region-wise angular flux by evaluating Eq. (2.21) for all regions
5. Compute scalar flux by evaluating Eq. (2.23) for all regions
6. Update 1-group source by evaluating Eq. (2.25) for all regions
7. Check for convergence, if not converged return to step 2.

Figure 2.2 – Iterative algorithm for the MOC solution of 1-group fixed source problem

For the algorithm of Figure 2.2, steps 2 through 5 perform the function evaluation shown in Eq. (2.24). In the 1-group fixed source problem, the external source, $Q_{ext,i,g}$, is assumed to be known which is essentially the source from fission and *in-scatter*, or scattering from group g' to group g, and it is a function of the scalar flux as shown in Eq. (2.26).

$$Q_{ext,i,g} = \frac{\chi_{i,g}}{k_{eff}} \sum_{g'=1}^{G} \nu\Sigma_{f,i,g'}\phi_{i,g'} + \sum_{\substack{g'=1 \\ g' \neq g}}^{G} \Sigma_{s0,i,g' \to g}\phi_{i,g'}. \qquad \textbf{Eq. (2.26)}$$

In Eq. (2.26), $1/k_{eff}$ is the eigenvalue of the system and must also be determined as a part of the solution. This is traditionally calculated using the power method, which is an iterative algorithm for finding the largest eigenvalue of a system that follows naturally from the source iteration scheme described thus far. Briefly, the general form of the eigenvalue value problem in reactor physics can be written in operator notation as:

$$\mathbf{T}\vec{\phi} = \frac{1}{k_{eff}}\mathbf{F}\vec{\phi}, \qquad \textbf{Eq. (2.27)}$$

where **F** represents the fission and **T** represents the streaming, absorption, and scattering of neutrons. Applying the power method to solve Eq. (2.27) results in the following iterative scheme:

$$\vec{\phi}^{(\ell+1)} = \mathbf{T}^{-1} \frac{1}{k_{eff}^{(\ell)}} \mathbf{F} \vec{\phi}^{(\ell)}, \qquad \textbf{Eq. (2.28)}$$

$$k_{eff}^{(\ell+1)} = \frac{\left\| \mathbf{F} \vec{\phi}^{(\ell+1)} \right\|_1}{\frac{1}{k_{eff}^{(\ell)}} \left\| \mathbf{F} \vec{\phi}^{(\ell)} \right\|_1}. \qquad \textbf{Eq. (2.29)}$$

This iterative scheme used in the second level of iteration for the eigenvalue, which is referred to as the *outer* iteration. For the outer iteration, denoted by the index ℓ, the total fission source is computed as shown in Eq. (2.30) and Eq. (2.26) is rewritten as shown in Eq. (2.31).

$$\psi_i^{(\ell)} = \frac{1}{k_{eff}^{(\ell)}} \sum_{g'=1}^{G} \nu\Sigma_{f,i,g'} \phi_{i,g'}^{(\ell)}, \qquad \textbf{Eq. (2.30)}$$

$$Q_{ext,i,g}^{(\ell,n)} = \chi_{i,g} \psi_i^{(\ell)} + \sum_{\substack{g'=1 \\ g' \neq g}}^{G} \Sigma_{s0,i,g' \to g} \phi_{i,g'}^{(\ell,n)}. \qquad \textbf{Eq. (2.31)}$$

The equation to update the eigenvalue based on the power method is shown in Eq. (2.32), where V_i is the region volume.

$$k_{eff}^{(\ell+1)} = \frac{\sum_{i=1}^{I} V_i \sum_{g'=1}^{G} \nu\Sigma_{f,i,g'} \phi_{i,g'}^{(\ell+1)}}{\sum_{i=1}^{I} V_i \psi_i^{(\ell)}}. \qquad \textbf{Eq. (2.32)}$$

The overall iterative procedure for solving the eigenvalue problem is shown in Figure 2.3 and is sometimes referred to as *source iteration*. In the source iteration technique described in this section, there is an inner iteration for the converging self-scattering and an outer iteration for the eigenvalue which is equivalent to the power iteration.

1. Guess initial k_{eff} and scalar flux.
2. Compute total fission source by evaluating Eq. (2.30) for all regions
3. Loop over all groups
   a. Compute 1-group source by evaluating Eq. (2.25) for all regions.
   b. Solve 1-group fixed source problem with algorithm in Figure 2.2
4. Update keff by evaluating Eq. (2.32) for all regions
5. Check for convergence, if not converged return to step 2.

Figure 2.3 – Iterative algorithm for the MOC solution of steady-state eigenvalue problem

This iteration scheme has the advantage of reducing memory usage, by allowing the transport method to only allocate data for a single group, with the exception of the boundary condition. In the loop over groups in Figure 2.3 the in-scatter source of Eq. (2.26) is also updated in a Gauss-Seidel fashion. This helps to improve convergence for reactor problems since most LWRs are thermal reactors and the physics of the slowing down source involves primarily the down-scatter of neutrons.

2.2 Discretization of the Characteristics

In Section 2.1 several kinds of discretizations were introduced for the different variables of the phase space. The discretization techniques for energy (the multi-group approximation) and space will not be discussed in this section since they are not specific to the MOC method, rather those discretizations are assumed and the focus will be on the specific discretization techniques required by the MOC method.

In the method of characteristics, the fundamental way that the problem is discretized is to choose a set of rays that traverse the problem domain to represent the flight paths (characteristic tracks) of the neutrons. This is illustrated in Figure 2.4 below. The end goal is to determine the segment lengths from each ray that pass through each discrete region, which are then used as the variable $s_{i,m,k}$ in the evaluation of Eq. (2.18), Eq. (2.20), and Eq. (2.21). In general, one may choose any set of rays so long as the intersection between the ray and the spatial region boundaries can be determined. At this point there are several design choices possible for the algorithm that performs the ray tracing.

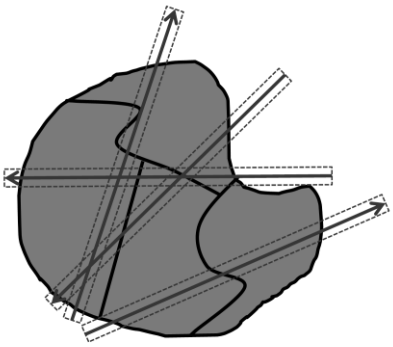


Figure 2.4 – Characteristic rays intersecting a set of discrete spatial regions

The first consideration is whether or not to store the ray tracing data (e.g. the segment lengths and mapping of ray segment index to region index). This information could be computed on the fly as a ray is swept during the transport sweep, or it can be stored. The criterion that is used to make this choice is to minimize computational time, and the tradeoff is essentially increased memory storage versus repetitive computation. The choice that is almost invariably made for any MOC implementation is to compute the ray tracing information once and then to store it. This has the benefit of decoupling the sweep algorithm from the ray trace algorithm, which allows each to be further optimized and developed essentially independently of the other. Another benefit of storing the ray tracing data is that during a normal calculation one may perform on the order of 1000 sweeps, and each sweep may involve iterating over tens or hundreds of millions of rays, so even the slightest overhead from the repeated computation of the ray tracing data will substantially increase the total computation time. However, the memory requirements for storage can become prohibitive, so other design choices for the algorithm must be made to address this issue. Despite the memory requirements for storing the ray tracing data, this is the approach used in this work.

The next consideration is the choice of rays, specifically, which directions of flight should be considered. Since the discrete ordinates approximation described in Section 2.1.3 is used, it logically follows that these should be used for the directions of flight for the rays. In order to obtain accurate solutions, the discrete directions of flight should be obtained from a quadrature that minimizes the error of the quadrature approximation for the integration of functions of angle. There are a myriad other

considerations that go into developing a good quadrature, but this will not be discussed in great detail. The usual terms in the transport equation that are functions of angle and are integrated over angle are the angular flux and scattering cross section.

Once the directions of flight are chosen, it is then necessary to set up rays for a given angle. Ideally, one would want a few rays from each angle to intersect every spatial region in the problem, but at the very least a single ray should intersect each region. One of the most common design choices here is to choose equally-spaced rays. The advantage of this choice is that it minimizes the need to store ray-dependent quantities such as the cross sectional area. Instead of storing the discrete cross sectional area of each characteristic ray in a problem, this quantity can simply be stored once for all rays or for all rays of a given angle. However, a potential disadvantage of uniform rays is one may place extra rays in regions that do not necessarily require them. This is illustrated in Figure 2.5 below.

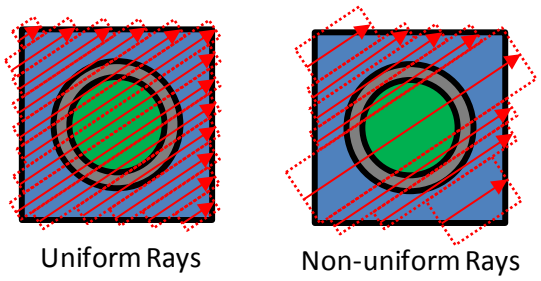


Figure 2.5 – Equally spaced and non-equally spaced characteristic rays

Since the segment volumes in a given region represent a numerical integration of the region volume, which will have some error, the segment lengths within a given region are renormalized so that they integrate the region volume exactly. This is shown in Figure 2.6 and Eq. (2.33).

$$\bar{s}_{i,m,k} = s_{i,m,k} \frac{V_i}{\sum_{k \in i} s_{i,m,k} \delta A_{m,k}}. \qquad \textbf{Eq. (2.33)}$$

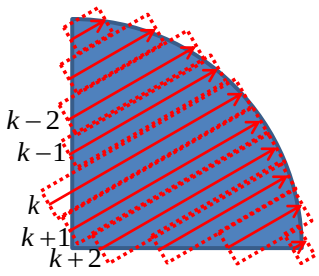


Figure 2.6 – Numerical integration of a region volume by ray segments

2.2.1 Modular Ray Tracing

In the design of the ray tracing algorithm, it is possible to take advantage of the fact that reactors generally have a high degree of regularity in their geometry. Considerable computational savings are possible by modeling only a small subdomain of the reactor that exhibits a unique geometry, and then constructing a ray tracing algorithm for the entire domain by replicating this information for the entire core. The technique for this has several names but is referred to as *modular ray tracing* here and is illustrated in Figure 2.7 below which has three *ray tracing modules* denoted as the black squares. The *modular rays*, depicted as blue lines, are defined only within the ray tracing module and connect at the ray tracing module boundaries. The *long ray* is shown by the red line and extends through the entire problem domain and consists of a particular sequence of modular rays.

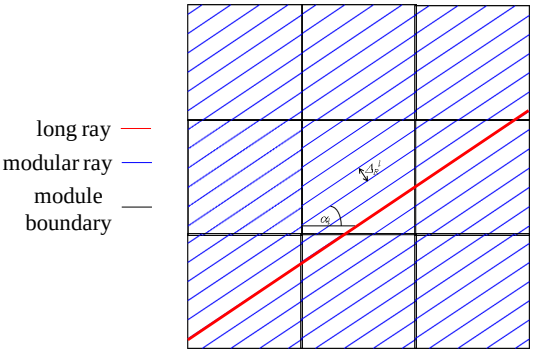


Figure 2.7 – Modular ray tracing concept in 2-D

It should be noted that the use of modular ray tracing introduces new requirements on the choice of the angles and also creates other subtle issues. The first requirement for modular ray tracing is that one be able to overlay a structured grid on their problem geometry. For the light water reactor problem this is a Cartesian grid, and

ideally will isolate the different unique geometries of the subdomains. The next requirement is that for rays with a given angle there must be an equal integer number of intersections of rays on opposing surfaces of a ray tracing module. This second requirement is basically satisfied by choosing to have equally spaced rays within a given angle. The computational advantages of modular ray tracing can be considerable. If modular ray tracing is not used, then it can increase the storage requirements of the ray tracing data by a factor up to as much as $O(10^7)$ for a problem using pin modular or quarter pin modular ray tracing that is approximately as large as a full core PWR. Should quarter assembly modular ray tracing be used, then the savings in memory requirements could be as much as $O(10^5)$ for a full core PWR.

The modular ray tracing technique has been previously used for 3-D MOC [32], [33], although a second, innovative technique for producing modular rays has been developed as a part of this work. The two algorithms basically differ in how the polar angle is determined. The result of this is that the first method is required to store twice as much ray tracing data when dealing with reflective boundary conditions. The new method also produces a finer ray spacings and consequently more rays, which can improve the accuracy of the solution but also increases computational work.

Once a modular geometric structure has been defined, the modular rays are determined by specifying a *desired* set of directions and ray spacing. In the work here a cuboid structure is used for the modular geometry with dimensions P_x, P_y, and P_z. The 3-D modular ray parameters of spacing Δ_r and Δ_z, azimuthal angle, α, and polar angle, θ, are determined from the inputs of a *desired* azimuthal angle, α_0, *desired* polar angle, θ_0, and *desired* ray spacing Δ_0. The first step in generating 3-D modular rays begins the same as it would for 2-D modular rays by only considering the *x*- and *y*- directions. The azimuthal angle, α, and radial ray spacing, Δ_r, are determined in this step from the following equations. For clarity, these terms are also illustrated in Figure 2.8.

$$N_x = \left| \frac{P_x}{\Delta_0} |\sin(\alpha_0)| \right|, \quad \textbf{(a)}$$

$$N_y = \left| \frac{P_y}{\Delta_0} |\cos(\alpha_0)| \right|, \quad \textbf{(b)}$$

Eq. (2.34)

$$\alpha = \arctan\left(\frac{P_y N_x}{P_x N_y} \right), \quad \textbf{(a)}$$

$$\Delta_r = \frac{P_x}{N_x} \sin(\alpha). \quad \textbf{(b)}$$

Eq. (2.35)

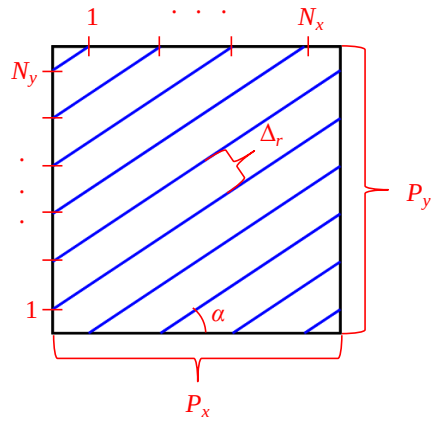


Figure 2.8 – Modular ray tracing parameters

In the first method, each 2-D line in Figure 2.8 is actually a plane in z-$\bar{\Omega}_{xy}$. This is illustrated in the part of Figure 2.9 above the black arrow. These planes are lined up in the order they are traversed along the direction $\bar{\Omega}_{xy}$, which is indicated by the red numbers in the same part of Figure 2.9. For example when the polar plane labeled "1" intersects the right edge, the next plane to be traversed is the one labeled "2" starting at the left edge, and so on until "10" returns to "1". Once the sequence of polar planes are ordered, it gives a new 2-D domain, which is the part of Figure 2.9 below the black

arrow. In this new 2-D domain the polar angle θ and axial ray spacing Δ_z are determined using Eq. (2.34) and Eq. (2.35) from above, where P_z is substituted for P_y and P_l is substituted for P_x.

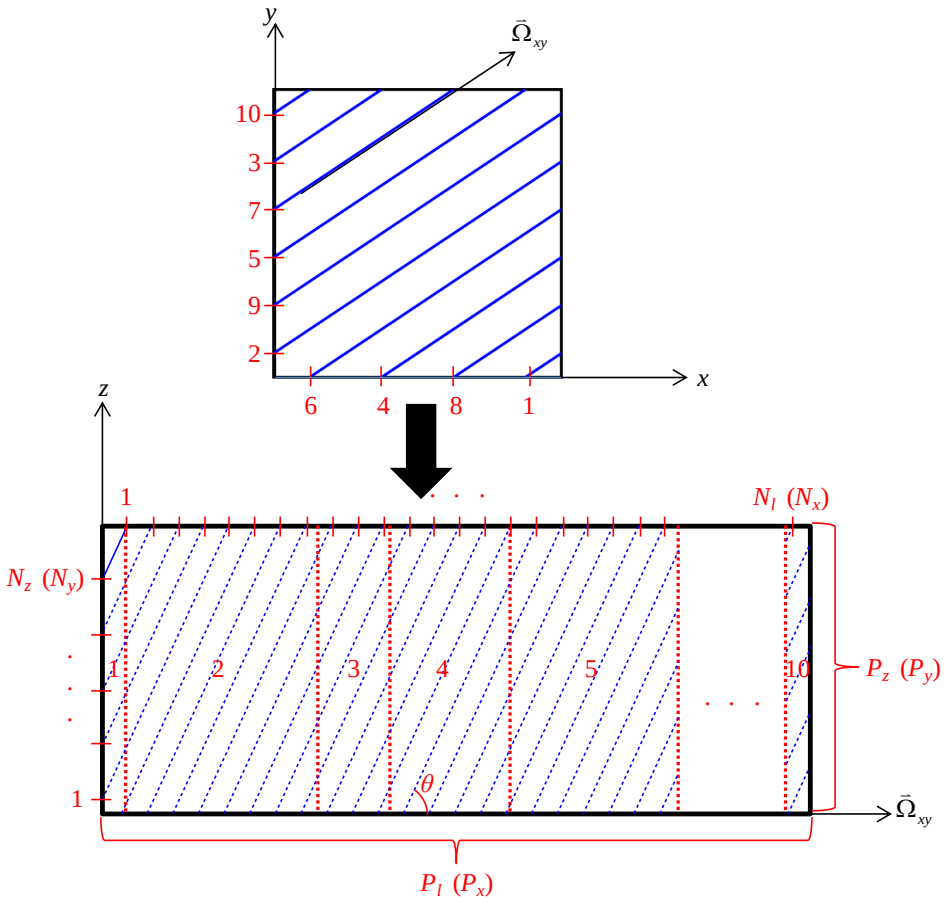


Figure 2.9 – Polar plane used to determine polar angle

A subtlety now arises in that one prefers not to have the modular rays intersect at the corners of the 2-D ray tracing modules. To avoid this, a shift parameter p is introduced, which determines how far from the corner the first ray is drawn. This is illustrated in Figure 2.10.

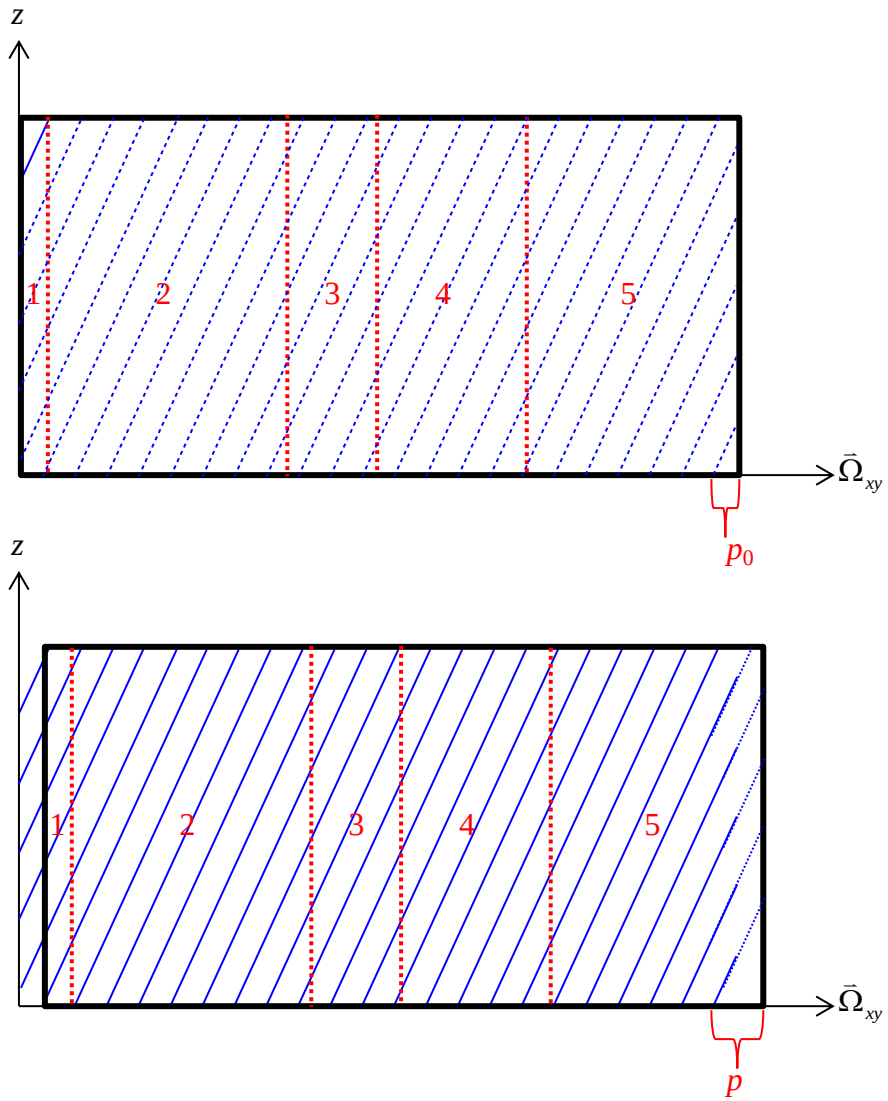


Figure 2.10 – Effect of shift parameter

Now, there is a further subtlety. The different directions will have slightly different shift parameters, so the problem that one encounters is when reflecting off a surface that changes the azimuthal direction the points of intersection for the rays with those two directions on that surface do not exactly line up. Therefore, one must store the forward and backward directions separately. Because each line does represent a forward and backward direction, one would hope to only be able to store directions over half the octants of the unit sphere rather, than all of them. This problem is illustrated in Figure 2.11.

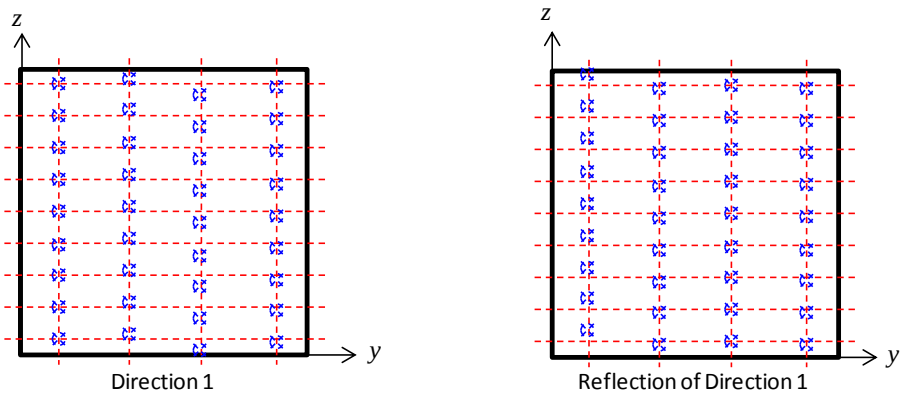


Figure 2.11 – Illustration of misaligned modular rays for reflecting directions

The new method developed for generating a set of modular rays as a part of this work avoids this problem by only considering the shortest polar plane. Noting that the rest of the polar planes for a given azimuthal direction have a length that is an integer multiple of the shortest plane means that the points of intersection along the *x*-*z* and *y*-*z* surfaces are the same for all polar planes and the forward and backward direction, so the ray tracing data of the four octants of the unit sphere in $0 \le \theta \le \pi$ must be stored. The method by which the polar angle is determined here is illustrated in the following figures.

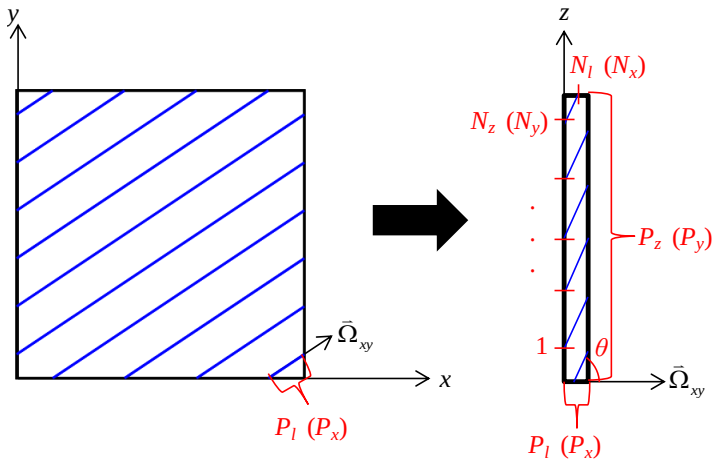


Figure 2.12 – Second way of determining the polar angle

Another subtlety in this method must also be noted. If the desired ray spacing is given preference instead of the desired polar angle, then the modular polar angle will be shifted substantially from the desired polar angle. This is due to the fact that the length of the polar plane is of the order of the desired ray spacing. This is illustrated in the Figure

2.13 below, where if the azimuthal angle is, $\alpha \approx \pi/4$; then polar angles in the range $\pi/4 \le \theta < \pi/2$ are essentially geometrically impossible if $\Delta_z \approx \Delta_r$ is desired.

This can introduce considerable problems with the angular quadrature as illustrated in Figure 2.14. Because robust methods of recomputing the quadrature weights have yet to be developed, and because the algorithm tends to cluster directions, it is currently not possible to obtain a good angular quadrature that is modular and has a spacing ratio near one using this method. The modular quadrature error is shown in Table 2.1 for the S_8 level-symmetric quadrature by comparing the analytic values of the moments of the unit sphere before and after the angular quadrature is modularized.

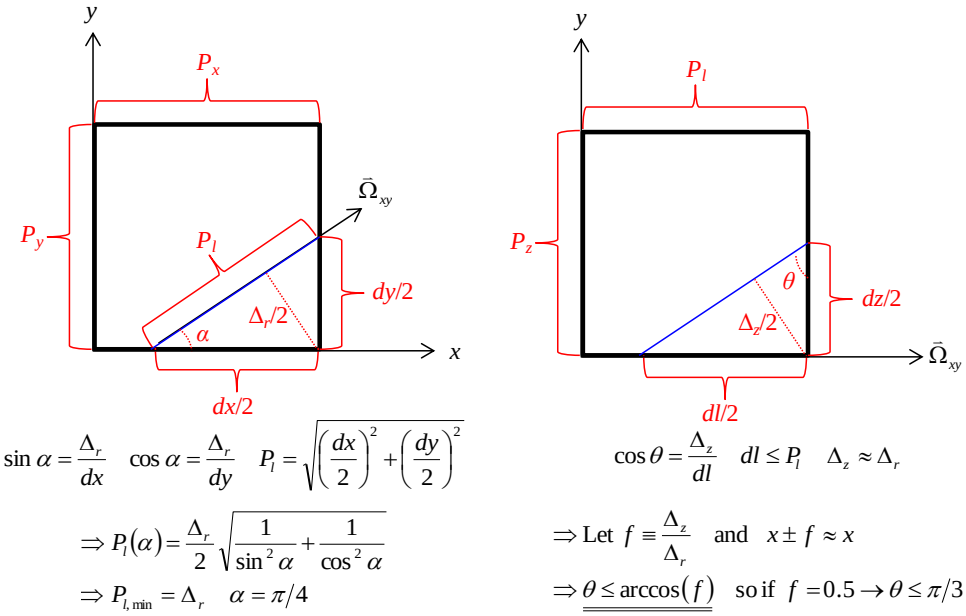
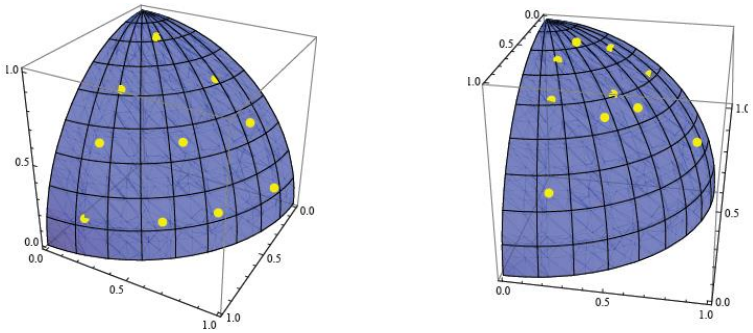


Figure 2.13 – Geometric limitation in the choice polar angle



Defined Quadrature (S_8) Modularized Quadrature

Figure 2.14 – Angular quadrature defined over an octant before and after modularization

Table 2.1 – S_8 quadrature errors of spherical harmonic moments

Moment	Defined Quadrature Rel. Error (%)	Modularized Quadrature Rel. Error (%)
$\int_{4\pi} 1 d\vec{\Omega} = 4\pi$	0.0	0.0
$\int_{4\pi} \Omega_x^2 d\vec{\Omega} = 4\pi/3$	0.0	34.0
$\int_{4\pi} \Omega_y^2 d\vec{\Omega} = 4\pi/3$	0.0	34.0
$\int_{4\pi} \Omega_z^2 d\vec{\Omega} = 4\pi/3$	0.0	68.1
$\int_{4\pi} \Omega_x^4 d\vec{\Omega} = 4\pi/5$	0.0	48.4
$\int_{4\pi} \Omega_y^4 d\vec{\Omega} = 4\pi/5$	0.0	48.4
$\int_{4\pi} \Omega_z^4 d\vec{\Omega} = 4\pi/5$	0.0	89.1
$\int_{4\pi} \Omega_x^2 \Omega_y^2 d\vec{\Omega} = 4\pi/15$	0.0	61.6
$\int_{4\pi} \Omega_x^2 \Omega_z^2 d\vec{\Omega} = 4\pi/15$	0.0	36.6
$\int_{4\pi} \Omega_y^2 \Omega_z^2 d\vec{\Omega} = 4\pi/15$	0.0	36.6
$\int_{4\pi} \Omega_x^6 d\vec{\Omega} = 4\pi/7$	0.0	54.5
$\int_{4\pi} \Omega_y^6 d\vec{\Omega} = 4\pi/7$	0.0	54.5
$\int_{4\pi} \Omega_z^6 d\vec{\Omega} = 4\pi/7$	0.0	35.5
$\int_{4\pi} \Omega_x^4 \Omega_y^2 d\vec{\Omega} = 4\pi/35$	0.0	74.2
$\int_{4\pi} \Omega_x^4 \Omega_z^2 d\vec{\Omega} = 4\pi/35$	0.0	7.6
$\int_{4\pi} \Omega_x^2 \Omega_y^4 d\vec{\Omega} = 4\pi/35$	0.0	74.2
$\int_{4\pi} \Omega_x^2 \Omega_z^4 d\vec{\Omega} = 4\pi/35$	0.0	73.1
$\int_{4\pi} \Omega_y^4 \Omega_z^2 d\vec{\Omega} = 4\pi/35$	0.0	7.6
$\int_{4\pi} \Omega_y^2 \Omega_z^4 d\vec{\Omega} = 4\pi/35$	0.0	73.1
$\int_{4\pi} \Omega_x^2 \Omega_y^2 \Omega_z^2 d\vec{\Omega} = 4\pi/105$	0.0	14.4

To address this issue, the way in which the polar angle is determined is modified so that an iteration is performed that adjusts the axial ray spacing Δ_z, typically reducing it, until the modular polar angle, θ, is found that most closely agrees with the desired polar angle, θ_0. This process can cause the number of rays for different directions to vary

considerably, but insures that the modularized angular quadrature is accurate. In Table 2.2, the improvement in the angular quadrature using this method is compared to the same quantities shown in Table 2.1 previously.

Table 2.2 – S_8 modular quadrature errors for preferred polar angle

Moment	Modularized Quadrature Rel. Error (%)
$\int_{4\pi} 1 d\vec{\Omega} = 4\pi$	0.0
$\int_{4\pi} \Omega_x^2 d\vec{\Omega} = 4\pi/3$	0.0
$\int_{4\pi} \Omega_y^2 d\vec{\Omega} = 4\pi/3$	0.0
$\int_{4\pi} \Omega_z^2 d\vec{\Omega} = 4\pi/3$	0.0
$\int_{4\pi} \Omega_x^4 d\vec{\Omega} = 4\pi/5$	0.0
$\int_{4\pi} \Omega_y^4 d\vec{\Omega} = 4\pi/5$	0.0
$\int_{4\pi} \Omega_z^4 d\vec{\Omega} = 4\pi/5$	0.0
$\int_{4\pi} \Omega_x^2 \Omega_y^2 d\vec{\Omega} = 4\pi/15$	0.0
$\int_{4\pi} \Omega_x^2 \Omega_z^2 d\vec{\Omega} = 4\pi/15$	0.0
$\int_{4\pi} \Omega_y^2 \Omega_z^2 d\vec{\Omega} = 4\pi/15$	0.0
$\int_{4\pi} \Omega_x^6 d\vec{\Omega} = 4\pi/7$	0.8
$\int_{4\pi} \Omega_y^6 d\vec{\Omega} = 4\pi/7$	0.8
$\int_{4\pi} \Omega_z^6 d\vec{\Omega} = 4\pi/7$	0.0
$\int_{4\pi} \Omega_x^4 \Omega_y^2 d\vec{\Omega} = 4\pi/35$	0.3
$\int_{4\pi} \Omega_x^4 \Omega_z^2 d\vec{\Omega} = 4\pi/35$	4.4
$\int_{4\pi} \Omega_x^2 \Omega_y^4 d\vec{\Omega} = 4\pi/35$	0.3
$\int_{4\pi} \Omega_x^2 \Omega_z^4 d\vec{\Omega} = 4\pi/35$	4.4
$\int_{4\pi} \Omega_y^4 \Omega_z^2 d\vec{\Omega} = 4\pi/35$	4.4
$\int_{4\pi} \Omega_y^2 \Omega_z^4 d\vec{\Omega} = 4\pi/35$	4.4
$\int_{4\pi} \Omega_x^2 \Omega_y^2 \Omega_z^2 d\vec{\Omega} = 4\pi/105$	0.0

Again the drawback of doing this method is that a larger number of rays are required. A comparison of the total number of rays that must be stored and traced is given in Table 2.3 for the S_8 quadrature and a desired spacing of 0.05 cm.

Table 2.3 – Comparison of modular rays requirements for different algorithms[1]

Angle	Number of Modular Rays for Algorithm 1 (Previous Research)	Number of Modular Rays for Algorithm 2 (This Research)
1	900	936
2	1038	1722
3	1038	1722
4	1050	1916
5	1134	1944
6	1050	1916
7	917	1571
8	1044	4122
9	1044	4266
10	917	1850
Total[1]	81056	87860

From the data in this table it can be concluded that it would be more computationally efficient to use the *algorithm 1* method because the total number of rays is less. Note that algorithm two is only storing half the rays, so the number that must be swept is actually double meaning that *algorithm 2* produces a set of rays that requires roughly 2x the FLOPS to perform a sweep compared algorithm 1.

If modular ray tracing is used, then there is a second step in the setup to determine the long ray information for the entire problem domain. This information essentially makes it possible to connect the modular rays in order to traverse through the entire domain. Once the ray tracing is completed, the MOC transport sweeps can be performed. As noted at the beginning of this section, the method used to perform a sweep depends to some extent on how the ray tracing is performed. The next section describes a few methods in which the MOC transport sweep can be performed.

[1] Algorithm 1 stores rays for the whole unit sphere and Algorithm 2 stores rays for just half the unit sphere, so the number of rays for each angle is multiplied by 8 and 4, respectively.

2.3 Overview of MOC Sweep Algorithms

This section presents an overview of a few of the MOC sweeping algorithms that have been successfully implemented. The description is focused on 2-D for ease of illustration. However, there is little that changes between a 2-D sweep and a 3-D sweep because in the MOC transport sweep, the fundamental serial operation is sweeping along all the segments in a long ray and evaluating Eq. (2.18) and Eq. (2.20). It is important to note that some implementations have been developed that formulate and solve a linear system for MOC [7], but these implementations have not received widespread application. The descriptions of the algorithms here are therefore limited to types of sweeps, rather than ways of solving a linear system. Because a linear system is not formed, the solution of the MOC transport method by sweeping presents a fundamentally different computational kernel than most other applications in scientific computing.

In the implementation of any sweeping algorithm, one of the first choices is the order in which the long rays are to be swept. Several techniques have been developed for performing a 2-D sweep; a few are explained here to highlight the important features characteristic of an efficient sweep algorithm. It is not practical to include descriptions of all approaches currently in use, since they are too numerous and some have not been documented. Nonetheless, the description here will highlight features that are common to all efficient sweep algorithms.

The first sweeping algorithm described is the very basic one, in which one assumes simply that there is some arbitrary set of rays that can be swept in an arbitrary manner. This is illustrated by Figure 2.15.

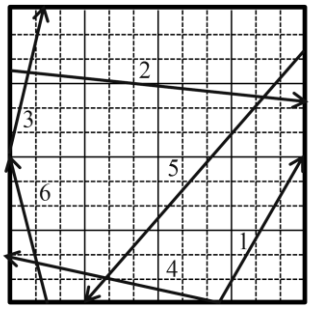


Figure 2.15 – Agnostic sweep algorithm

Because the ordering of rays in the sweep is arbitrary, it is most likely that this sweep is probably not optimal. The memory requirements for an arbitrary sweep are typically large because the boundary condition would need to be stored for the end of each ray, and the ray tracing data would be stored separately for the forward and backward direction. If modular ray tracing was not used, then an exact specular reflective boundary conditions cannot be treated, and some approximation will be required at the boundary which could affect the solution accuracy. Furthermore, it is difficult to conclude anything about the efficiency of the convergence of this type of sweep, since some rays could be swept and then rays that might update those boundary conditions will not be swept until sometime later. Additionally, it is expected that an arbitrary sweep algorithm will not have good cache coherency. A long ray may have its ray tracing data loaded into cache while it is swept for one direction, and then replaced by data for rays in another direction. At some later time, the data for long ray would have to be reloaded (albeit in the reverse order) in order to sweep in the reverse direction. Obviously, from the cache efficiency standpoint, a more efficient sweep method would be to sweep both forward and backward directions of a ray while the data resides in cache.

The oldest sweep algorithm is the cyclic method [30], which provides a method of sweeping the rays that is basically motivated by two fundamental observations about the problem. The first condition is that the problem has reflective boundary conditions. The second condition is that the ray tracing data is built modularly. A reflective boundary condition is typical of single assembly 2-D calculations, whereas full core calculations have at least some vacuum boundary conditions. Given these two conditions, the first obvious way to improve the efficiency of the ray sweeping is to sweep over a cycle or *loop* of rays. The benefits of sweeping a loop, rather than from one surface to another, is a reduction in memory requirements for storing the angular flux boundary condition, since it is computed on the fly at each surface. This will improve convergence because the angular flux boundary condition is updated in a Gauss-Seidel like fashion. The figure below illustrates how a cyclic ray is swept.

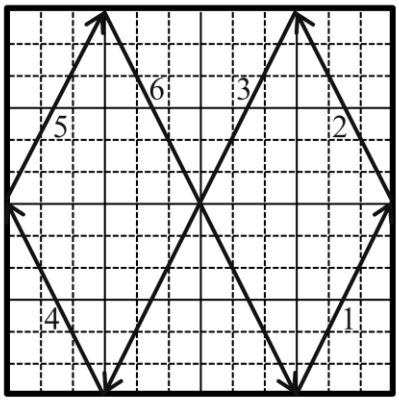


Figure 2.16 – Cyclic ray sweep algorithm

A potential disadvantage to this kind of sweeping is poor cache coherency. This is because each ray represents two directions of flight, so its ray tracing data is simply reversed for the forward and backward direction. If the cyclic rays are traced first for the forward direction and then, for the backward direction, the ray tracing data is essentially loaded into and out of the cache twice, which can diminish the performance and increase the time to perform a sweep by nearly a factor of two [42].

This leads to the second idea of how to sweep rays, which probably seems more obvious to the procedural programmer, and that is to loop over the angles and then within each angle loop over all the long rays. The advantage of this kind of sweep is that it becomes easier to order all the associated data structures to have good cache coherency. If modular ray tracing is being performed, then the construction of the long ray can be done once and swept for both the forward and backward direction (e.g. a bi-directional sweep), further improving performance through better cache coherency. The figure below illustrates how the rays can be swept sequentially.

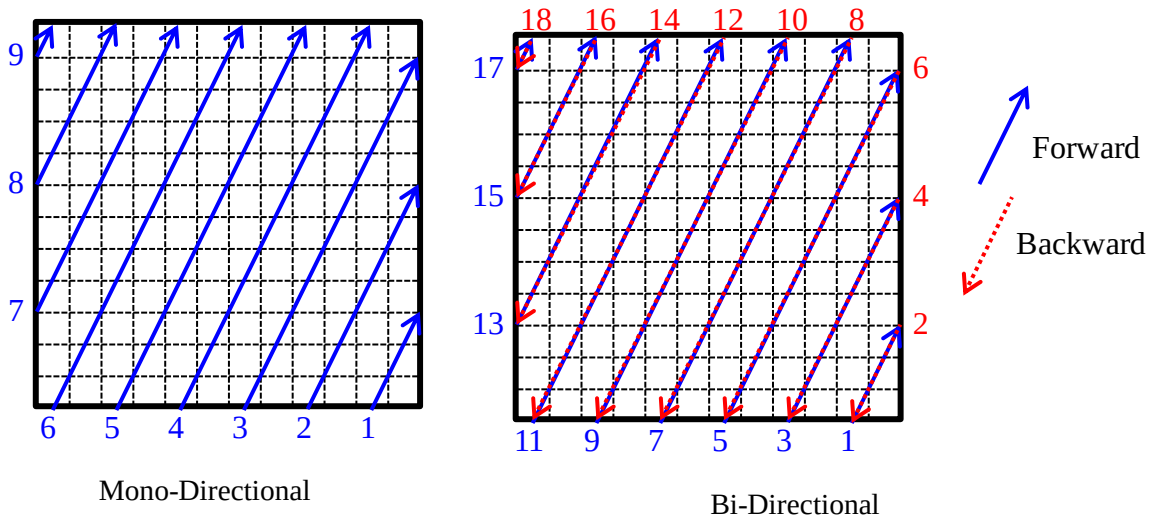


Figure 2.17 – Sequential sweep algorithm

The disadvantages for this kind of sweep ordering are basically the opposite of the advantages of the cyclic sweep. Namely, there is increased memory storage when there are reflective or periodic boundary conditions, and the convergence of the angular flux boundary condition will be more like Gauss-Jacobi than Gauss-Seidel.

Thus, the two approaches for sweeping the rays, either cyclically or a sequentially, have opposing advantages and disadvantages, and the most efficient algorithm would be an optimal compromise of the two methods. This type of algorithm is illustrated below.

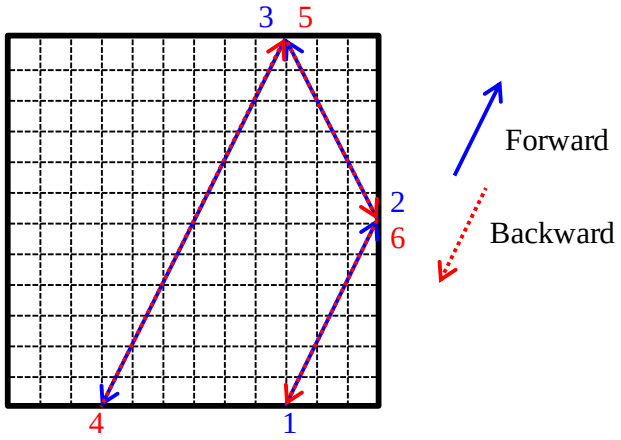


Figure 2.18 – Bi-directional surface-cyclic sweep algorithm

The surface-cyclic sweep algorithm is better able to allow for cache coherency than the cyclic algorithm, most notably because the total track length will be smaller and

thus more likely to fit within the lowest level of cache. This has the benefits of the sequential bi-directional sweep, and if there are reflective boundary conditions, then it also has the benefit of having to store fewer elements of the angular flux boundary condition. Also, since these elements will only be stored for one surface of the domain, it therefore has similar advantages as the full cyclic sweep.

Of the various sweep algorithms discussed here, the sequential bi-directional sweep is chosen as the basis for the optimizations and kernel implementation described in the next chapter and Chapter 4, which describes the parallelization method. The reason for choosing this algorithm is not readily apparent from the discussion in this chapter, but the relative merits of the sequential bi-directional sweep should become clear in the following chapters.

Chapter 3

3-D MOC KERNEL IMPLEMENTATION

This chapter describes how the equations derived in Chapter 2 are implemented within a programming model. The goal of this chapter is to establish the basis for a serial 3-D MOC kernel that can then be parallelized. Details are also given about some up-front optimizations that have been used previously in 2-D MOC kernels and leveraged in the 3-D MOC kernel. The first section of this chapter shows how the equations of Section 2.1 can be manipulated algebraically to minimize the number of floating point operations (FLOPs). Section 3.2 describes how the exponential function that must be evaluated can be pre-tabulated, to further improve the computational performance of the kernel. Finally, the detailed step-by-step procedure of the 3-D MOC kernel is presented.

3.1 Algebraic Optimization of Discretized MOC Equations

For the actual implementation of a MOC kernel there is some algebraic optimization that may be done to reduce the number of floating point operations. The approach described here is well known in the research community, and is shown here in detail so that there is no ambiguity about the order of operations in the kernel. The technique is based on the idea of using algebra to eliminate terms from some equations which essentially removes operation in inner loops to the outer loops. This approach makes several assumptions about the data available and the approximations used to derive the discrete equations; these are consistent with the derivation of Section 2.1. The derivation begins by repeating the relevant discretized equations from Section 2.1 solved by the MOC kernel:

$$\varphi_{i,g,m,k}^{out} = \varphi_{i,g,m,k}^{in} \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right) + \frac{q_{i,g,m}}{\Sigma_{t,i,g}}\left[1 - \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right)\right] \quad \textbf{Eq. (2.18)}$$

Eq. (2.18) – MOC transmission equation

$$\tilde{\varphi}_{i,g,m,k} = \frac{\varphi_{i,g,m,k}^{in} - \varphi_{i,g,m,k}^{out}}{\Sigma_{t,i,g} s_{i,m,k}} + \frac{q_{i,g,m}}{\Sigma_{t,i,g}} \quad \textbf{Eq. (2.20)}$$

Eq. (2.20) – Equation for segment average angular flux

$$\overline{\varphi}_{i,g,m} = \frac{\sum_k \tilde{\varphi}_{i,g,m,k} s_{i,m,k} \delta A_{m,k}}{\sum_k s_{i,m,k} \delta A_{m,k}} \quad \textbf{Eq. (2.21)}$$

Eq. (2.21) – Equation for region average angular flux

$$\phi_{i,g} = \sum_k w_m \overline{\varphi}_{i,g,m} \quad \textbf{Eq. (2.23)}$$

Eq. (2.23) – Equation for region average scalar flux

Traditionally, for reactor applications it is sufficient for the MOC kernel to just update the scalar flux in all regions. Since this is ultimately the quantity of interest to be updated for the solution, the following steps will optimize the calculation for the scalar flux. It is noted that the evaluation of the transmission equation represents the operation performed in the inner most loop.

First is the definition of an intermediate quantity taken to be the angular flux difference across a segment.

$$\varphi_{i,g,m,k}^{d} \equiv \varphi_{i,g,m,k}^{in} - \varphi_{i,g,m,k}^{out}. \quad \textbf{Eq. (3.1)}$$

This implies the following:

$$\varphi_{i,g,m,k}^{out} = \varphi_{i,g,m,k}^{in} - \varphi_{i,g,m,k}^{d}. \quad \textbf{Eq. (3.2)}$$

Now inserting Eq. (2.18) into Eq. (3.1) so it can be rewritten in terms of $\varphi_{i,g,m,k}^{in}$ gives:

$$\varphi^{d}_{i,g,m,k} = \varphi^{in}_{i,g,m,k} - \left(\varphi^{in}_{i,g,m,k} \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right) + \frac{q_{i,g,m}}{\Sigma_{t,i,g}} \left[1 - \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right)\right] \right)$$

$$\Rightarrow \varphi^{d}_{i,g,m,k} = \varphi^{in}_{i,g,m,k} \left[1 - \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right)\right] - \frac{q_{i,g,m}}{\Sigma_{t,i,g}} \left[1 - \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right)\right]$$

$$\varphi^{d}_{i,g,m,k} = \left(\varphi^{in}_{i,g,m,k} - \frac{q_{i,g,m}}{\Sigma_{t,i,g}} \right) \left[1 - \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right)\right] \qquad \textbf{Eq. (3.3)}$$

Eq. (3.2) and Eq. (3.3) then replace Eq. (2.18) and have removed at least 1 FLOP from the inner-most loop. Another useful definition is to define a "reduced" fixed source for the region given as:

$$\bar{q}_{i,g} \equiv \frac{q_{i,g}}{\Sigma_{t,i,g}}, \quad q_{i,g} \leftarrow q_{i,g,m}\,. \qquad \textbf{Eq. (3.4)}$$

It is noted here that by dropping the angle index, an isotropic source has also been assumed, so this will not apply to MOC kernels that explicitly treat sources that are higher order in angle or space. Substituting Eq. (3.4) into Eq. (3.3) allows $\bar{q}_{i,g}$ to be pre-computed removing a division operation from the inner-most loop and also removing the angle dependence of the term. Rewriting Eq. (3.3) with Eq. (3.4) gives:

$$\varphi^{d}_{i,g,m,k} = \left(\varphi^{in}_{i,g,m,k} - \bar{q}_{i,g}\right) \left[1 - \exp\left(-\Sigma_{t,i,g} s_{i,m,k}\right)\right] \qquad \textbf{Eq. (3.5)}$$

Next, the focus is turned to Eq. (2.20) and Eq. (2.21). Substituting Eq. (3.1) and Eq. (3.4) into Eq. (2.20) gives:

$$\tilde{\varphi}_{i,g,m,k} = \frac{\varphi^{d}_{i,g,m,k}}{\Sigma_{t,i,g} s_{i,m,k}} + \bar{q}_{i,g}\,. \qquad \textbf{Eq. (3.6)}$$

Substituting the above into Eq. (2.21) and rearranging leads to:

$$\bar{\varphi}_{i,g,m} = \frac{\sum_k \left(\frac{\varphi^d_{i,g,m,k}}{\Sigma_{t,i,g} s_{i,m,k}} + \bar{q}_{i,g} \right) s_{i,m,k} \delta A_{m,k}}{\sum_k s_{i,m,k} \delta A_{m,k}},$$

$$\Rightarrow \bar{\varphi}_{i,g,m} = \frac{\sum_k \left(\frac{\varphi^d_{i,g,m,k}}{\Sigma_{t,i,g} s_{i,m,k}} s_{i,m,k} \delta A_{m,k} \right)}{\sum_k s_{i,m,k} \delta A_{m,k}} + \frac{\sum_k \bar{q}_{i,g} s_{i,m,k} \delta A_{m,k}}{\sum_k s_{i,m,k} \delta A_{m,k}}$$

$$\Rightarrow \bar{\varphi}_{i,g,m} = \frac{\sum_k \left(\varphi^d_{i,g,m,k} \delta A_{m,k} \right)}{\Sigma_{t,i,g} \sum_k s_{i,m,k} \delta A_{m,k}} + \bar{q}_{i,g} \frac{\sum_k s_{i,m,k} \delta A_{m,k}}{\sum_k s_{i,m,k} \delta A_{m,k}}$$

$$\bar{\varphi}_{i,g,m} = \frac{\sum_k \left(\varphi^d_{i,g,m,k} \delta A_{m,k} \right)}{\Sigma_{t,i,g} \sum_k s_{i,m,k} \delta A_{m,k}} + \bar{q}_{i,g}. \qquad \textbf{Eq. (3.7)}$$

The next operation is based on two observations about the ray tracing. In the case of modular ray tracing, and in general when there is a uniform ray spacing, the segment cross sectional area will only be a function of angle, and therefore:

$$\delta A_m \leftarrow \delta A_{m,k}. \qquad \textbf{Eq. (3.8)}$$

Additionally, the ray segments must be adjusted to preserve region volumes, so the following should always be true:

$$V_i = \sum_k{}' s_{i,m,k} \delta A_{m,k}. \qquad \textbf{Eq. (3.9)}$$

Substituting the above into Eq. (3.7) then gives:

$$\bar{\varphi}_{i,g,m} = \frac{\delta A_m \sum_k \varphi^d_{i,g,m,k}}{V_i \Sigma_{t,i,g}} + \bar{q}_{i,g}. \qquad \textbf{Eq. (3.10)}$$

Finally, inserting Eq. (3.10) into Eq. (2.23) gives:

$$\phi_{i,g} = \frac{\sum_m w_m \delta A_m \sum_k \varphi^d_{i,g,m,k}}{V_i \Sigma_{t,i,g}} + \bar{q}_{i,g} \sum_m w_m . \qquad \textbf{Eq. (3.11)}$$

The figure below outlines the procedure in which the sweep can now be performed more efficiently. For notational convenience, the following expressions are also defined:

$$\hat{\varphi}_{i,g,m} = \sum_k \varphi^d_{i,g,m,k} , \qquad \textbf{Eq. (3.12)}$$

$$\hat{\phi}_{i,g} = \sum_m w_m \delta A_m \hat{\varphi}_{i,g,m} , \qquad \textbf{Eq. (3.13)}$$

$$\phi_{i,g} = \frac{\hat{\phi}_{i,g}}{V_i \Sigma_{t,i,g}} + (4\pi)\bar{q}_{i,g} . \qquad \textbf{Eq. (3.14)}$$

For group *g*:

1. Evaluate Eq. (3.4) for all regions
2. Loop over all angles
    a. Loop over all long rays in angle *m*
        i. Loop over all segments in long ray
            1. Evaluate Eq. (3.5) in forward direction
            2. Accumulate $\varphi^d_{i,g,m,k}$ into temporary for Eq. (3.12) for forward direction.
            3. Evaluate Eq. (3.5) in backward direction
            4. Accumulate $\varphi^d_{i,g,m,k}$ into temporary for Eq. (3.12) for backward direction.
        ii. Accumulate $w_m \delta A_m \hat{\varphi}_{i,g,m}$ into temporary for Eq. (3.13)
3. Evaluate Eq. (3.14) for all regions

Figure 3.1 – Iterative sequence for optimized equations

By rewriting the equations this way the overall number of FLOPs required to evaluate the MOC equations is reduced by approximately 70%.

3.2 Tabulated Linear Interpolation of the Exponential Function

Another well-known optimization of MOC kernels is to tabulate the exponential function or more specifically, 1-exp(*x*), since this is the term that appears in Eq. (3.5). In previous work [43], the tabulation error and speedup from using the table with various forms of interpolation were investigated. This work is replicated here and similar results are obtained. For the evaluation of the exponential function, the table with linear interpolation is used, since this should be the most efficient type of table evaluation as suggested in [43]. In the characteristics equation the argument for the exponential function is $-\Sigma_{t,i,g} s_{i,m,k}$. The total cross section and segment length will always be positive, so the argument to the exponential function will always be less than or equal to zero. The value of the exponential function for negative real numbers is shown next:

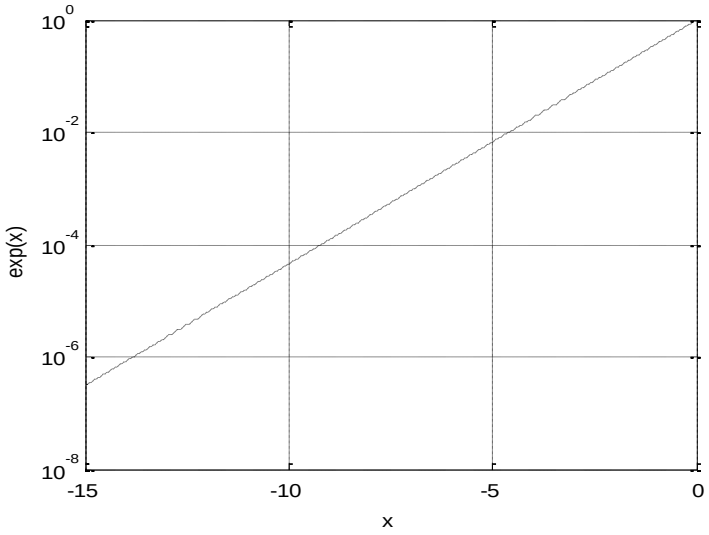


Figure 3.2 – Range of values of exponential function in characteristics problem

In this figure it is noted that the value of the exponential function changes very little in magnitude (< 1.e-4) beyond -10.0. Therefore, the range of interpolation is chosen to be $x \in [-10,0]$. The interpolation error can also be predetermined as a function of the number of intervals in the table. Generally, as small a table as possible is best so that it can remain completely in the L1 cache, which has the fastest memory access times. The error introduced from the linear interpolation table lookup was evaluated numerically and

bounded by theory; these errors are shown in Figure 3.3, which is consistent with the results of [43].

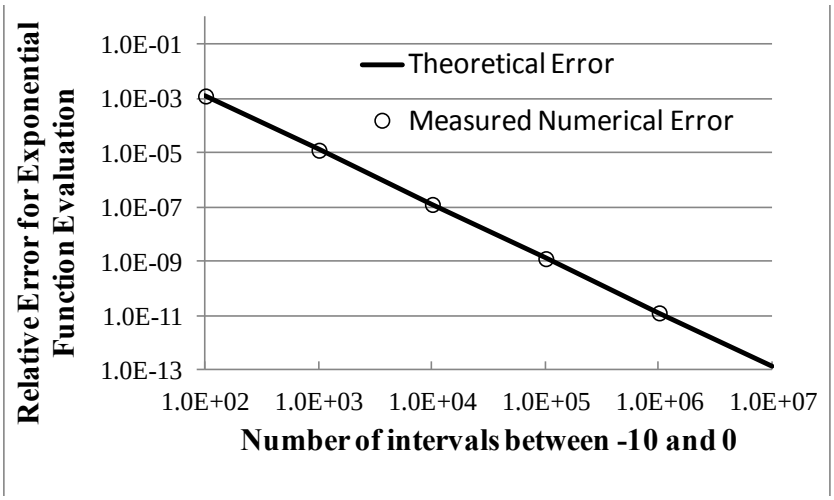


Figure 3.3 – Error of linear interpolation of exponential function

The performance gain of the table evaluation over evaluation of the intrinsic function is shown in Table 3.1. This performance gain can be attributed to trading floating point operations with memory accesses. As long as the table is capable of residing in fast memory, the number of processor clock cycles for a memory access is of the same order as that required to perform a FLOP, and therefore a performance improvement will be achieved. This was verified using the measurement system described in Chapter 5, with the GNU 4.6.3 Fortran compiler and `-Ofast` optimizations. The test consisted of 300 random accesses to the table and was averaged over 100,000 samples and by using the table the number of FLOPs was reduced by roughly an order of magnitude, while a 76% reduction in time was observed.

Table 3.1 – Performance gain from table evaluation of the exponential function

	Intrinsic Exponential Function	Table look up with Linear Interpolation (1000 intervals)
Time	1.42 s	0.334 s
FLOPS	1.051e9	2.414e8

3.3 Anatomy of the 3-D MOC Kernel

In this section, the detailed algorithm for the serial 3-D MOC kernel is described. This serial algorithm is the basis for the extension to a parallel algorithm in Chapter 4 and the performance analysis of Chapter 5 and Chapter 6. Functionally, the kernel performs the operation shown by Eq. (3.15). This is to serve as the transport sweep at the heart of the overall eigenvalue iteration, as shown in Figure 3.4, which is valid for any transport method. The procedure by which the MOC kernel performs the transport sweep is shown in Figure 3.5.

$$\left\{\vec{\phi}_g^{(\ell+1)}, \vec{\varphi}_g^{out(\ell+1)}\right\} = f\left(\vec{\varphi}_g^{in(\ell)}, \bar{\bar{q}}_g^{(\ell)}\right) \qquad \textbf{Eq. (3.15)}$$

```
While not converged
   1. Compute total fission source
   2. Loop from group 1 to G
         a. Compute fission source for group g
         b. Add in-scatter source to fission source
         c. Add within-group scattering to source
         d. Evaluate Eq. (3.4)
         e. Evaluate Eq. (3.15)
   3. Update k_eff
   4. Check if solution is converged
```

Figure 3.4 – Basic algorithm for source iteration

1. Loop over all angles
    a. Loop over all long rays in angle
        i. Order the modular ray data for a complete long ray
        ii. Evaluate the exponential function for all segments in the long ray
        iii. Load incoming boundary conditions for each end of ray
        iv. Loop over all segments in the long ray
            1. Evaluate Eq. (3.5) in forward direction
            2. Accumulate $\varphi^{d}_{i,g,m,k}$ into temporary for Eq. (3.12) for forward direction.
            3. Evaluate Eq. (3.5) in backward direction
            4. Accumulate $\varphi^{d}_{i,g,m,k}$ into temporary for Eq. (3.12) for backward direction.
        v. Store outgoing boundary conditions for each end of ray
    b. Accumulate $w_m \delta A_m \hat{\varphi}_{i,g,m}$ into temporary for Eq. (3.13)
    c. Update all incoming boundary conditions
2. Evaluate Eq. (3.14) for all regions

Figure 3.5 – 3-D MOC kernel serial algorithm

The first step in this algorithm is to use the modular ray data to determine the spatial region index, segment length, and total cross section sequentially by segment for a single long ray from one end of the ray to the other. This data is stored into local 1-D arrays. Next, the segment length and cross section are used to evaluate the exponential function for all segments along the ray. Then a loop is performed over all the segments on the long ray, and the MOC equations, Eq. (3.3) and Eq. (3.12) are evaluated. Once all the long rays have been swept within a given angle, the sums accumulated in each region for Eq. (3.12) are scaled and accumulated based on Eq. (3.13). The final step is to evaluate Eq. (3.14) for all regions once the loop over all angles is completed. This is essentially steps 2 and 3 in Figure 3.1. Each of the steps of Figure 3.5 can be thought of as a different operational component, or microkernel, of the overall 3-D MOC kernel. These components are given the following monikers as shown in Table 3.2 for use in discussion and equations in subsequent chapters.

Table 3.2 – Components of 3-D MOC kernel

Algorithm Step	Name of micro-kernel	Description
i	*build*	Order the modular ray data sequentially to construct data for a single long ray
ii	*exp*	Evaluate exponential functions for $-\Sigma_{t,i,g} s_{i,m,k}$ values of each segment in long ray.
iii and v	*BC*	Load and store long ray boundary conditions in global memory
iv	*MOC*	Evaluate MOC equations Eq. (3.5) and Eq. (3.12)
b	*scal*	Scale region-wise flux sums by angular weight using Eq. (3.13)
c	*BCUp*	Update all incoming boundary conditions
2	*flux*	Compute the scalar flux using Eq. (3.14)

Chapter 4

PARALLEL ALGORITHM FOR SOLVING 3-D METHOD OF CHARACTERISTICS

In this chapter the parallelization is described for the 3-D MOC kernel shown in Figure 3.5 from the previous chapter. The general approach is to perform domain decomposition on each of the variables of the phase space in the transport equation. Additionally, some parallelism is implemented, which is specific to the discretization required for the MOC solution of the transport equation. Before the approach to the parallel decomposition is described in detail, some basic concepts of parallel computing are introduced. Finally an overall summary of the parallel algorithm is given.

4.1 Basics of Parallel Computing

The primary models of architecture in parallel computing are *shared* and *distributed memory*. In the shared memory model, it is assumed that all parallel processes have access to the same memory. The most common implementation of this model in scientific computing is OpenMP [44] and generally the term *thread* is used to refer to the different parallel processes, although there are other implementations that are also commonly used. In the distributed memory model each parallel process is assumed to have its own memory separate from other processes, and if different processes require the data from another process, it is communicated via message passing over a network. The most common implementation of the distributed memory model is MPI [45], although there are other implementations of this model as well, such as co-array Fortran. There is also a third model called the *hybrid model*, which is a combination of distributed and shared

memory models, where a small set of processors may have shared memory and be connected to other distributed sets of processors.

Most of the high end parallel computers today use the distributed memory model or a hybrid model, and it appears that the hybrid model will continue to be characteristic of high performance computing architectures through exascale computing, although the shared memory environments on distributed processes may undergo significant architectural changes [46].

Other concepts of parallel computing that are important to understand are those of *partitioning* and *load balancing*. Partitioning is the algorithm by which a domain is decomposed. In a decomposed problem, the load balance refers to the amount of work given to each process. Generally in parallel algorithms there are synchronization points, and if some processors have a disproportionate amount of work, then this can cause other processors to wait for those with more work, creating a bottleneck. Therefore, achieving a good load balance helps to maximize the performance of almost any parallel algorithm. The load balance is determined by the partitioning algorithm, so in parallel algorithms the partitioning of the domain is often a critical component to achieving good performance.

4.2 Decomposition of the Spatial Domain

The spatial decomposition is employed using a distributed memory model. This is necessary to alleviate the memory burden required by full core problems [7]. The basic idea behind the spatial decomposition is to divide the full core problem into enough spatial domains such that each domain is of a size that is manageable for a single process. In order to implement spatial domain decomposition, the principal issue is the overhead in communicating the boundary condition information across the spatial subdomain boundaries. This is evident in Eq. (2.18) and illustrated in Figure 4.1 for a 2-D domain.

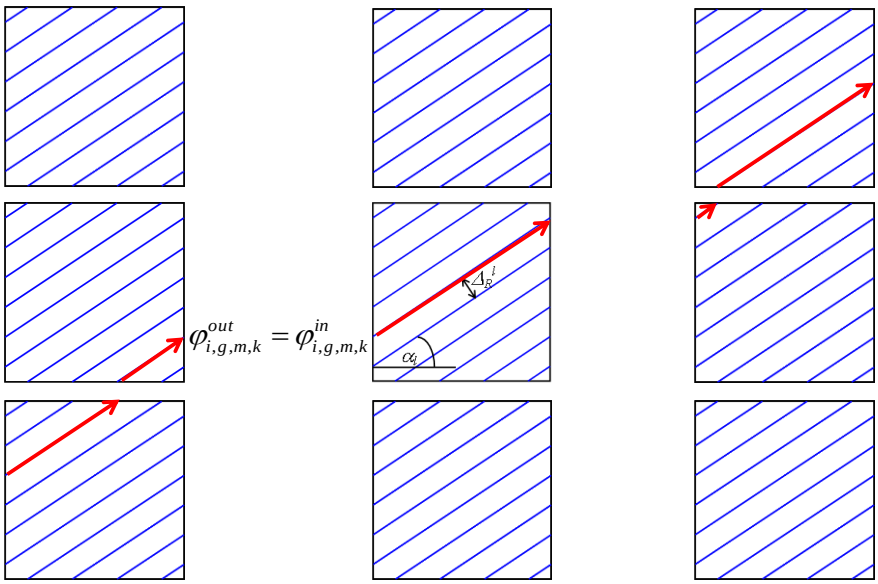


Figure 4.1 – 2-D sweep with domain decomposition

Unless approximations are introduced, the characteristic rays must be directly connected across these boundaries, in order to transfer the interface angular fluxes. Furthermore, because the incoming boundary condition must be stored on all surfaces shared between parallel subdomains the bi-directional sequential sweep algorithm illustrated in Figure 2.18 would be the preferred sweep algorithm in a given spatial subdomain. Another detail of this decomposition is the order by which the subdomains may be traversed during a single iteration. In the work here, the subdomains are allowed to perform their local sweeps simultaneously with all other subdomains. This is essentially equivalent in concept to a block Jacobi type iteration scheme. The KBA wave front algorithm [16] could also be used instead, which would have better convergence properties, but the block Jacobi scheme was chosen to be studied first since it will maximize parallelism. By allowing the subdomains to solve their local problems simultaneously, it fundamentally changes the iteration scheme, and therefore the rate of convergence when compared to the serial iteration. Interior subdomains will not have the same boundary condition as the serial problem until it has been communicated through all other subdomains between the interior subdomain and the problem boundary. This is also illustrated by Figure 4.1.

When choosing the optimal spatial decomposition, it is important to consider the advantages provided by modular ray tracing, which imposes a virtual structured grid,

which is Cartesian in the case of LWRs. A direct connection of rays at the interfaces of this structured grid is also guaranteed, so there are no further approximations that are introduced. This was realized in previous work [47] on the parallelization of 2-D MOC solvers for full core problems where the modular geometry grid for the modular ray tracing was used as the basis for the spatial decomposition. Furthermore, the spatial mesh description within each modular geometry, or element of the grid, is relatively unrestricted and there is no significant limitation as to the complexity of the geometry that may be modeled. By choosing a decomposition centered on modular geometry, different locations in the grid are likely to have the same geometric description, and therefore each domain is likely to have the same amount of work, which facilitates load balance. Second, algorithms for partitioning a structured Cartesian modular grid are simpler to develop and implement. For these reasons the modular grid is ideal for use in decomposing the problem.

There are other considerations for the spatial decomposition that should help to achieve the best possible load balancing and minimization of communication time. For example, it is important for each partitioned subdomain to have a near optimal surface to volume ratio, since the amount of work is proportional to the volume, and the amount of communication is proportional to the subdomain surface area. Spatial subdomains are also restricted to only having one neighboring subdomain on each face, which minimizes the number of messages that must be used to communicate the boundary conditions.

A partitioning algorithm was developed based on the *Morton*, or *Z-order*, space filling curve [48]. Several types of space-filling curves exist, and the Morton curve was chosen primarily because of the simplicity with which it could be implemented. The Morton curve is pictured below in Figure 4.2 for 2-D and 3-D. Space filling curves usually have the advantage of being naturally hierarchical, and thus they can be easily represented by a tree data structure and recursive algorithms. They also insure that at any level within the hierarchy, the numbering will be continuous within each subdomain. To accomplish the partitioning, a tree data structure similar to a *k-d* tree or partially filled oct-tree was implemented and for the purposes of discussion it will be referred to as a Z-tree.

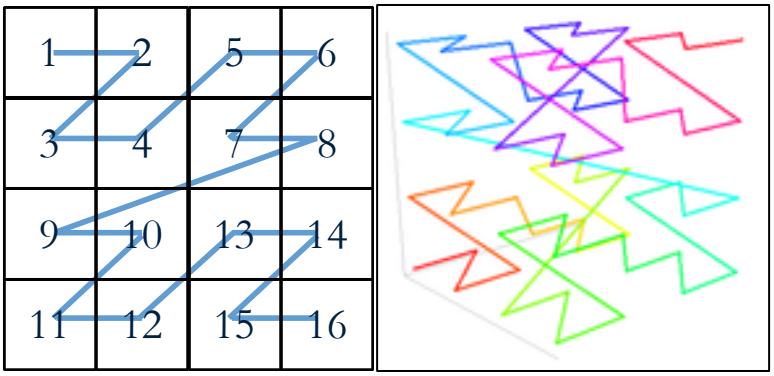


Figure 4.2 – Morton (Z-order) curve of the 2nd order in 2-D (left) and 3-D (right)[2]

The Z-Tree essentially determines a global indexing for the modular geometry mesh such that two domains that are geometrically close also have index values that are relatively close numerically. Furthermore, the indexing within a subdomain will be continuous. The other thing to note about the space filling curves is they are typically only defined in Cartesian space on grids with dimensions of 2^n.

Since the generic reactor problem is not likely to have this kind of grid, equations for these curves should not be used directly, since it will create a non-contiguous numbering. The Z-tree generates the indexing by taking grid dimensions in *x*-, *y*-, and *z*-dimensions and dividing this grid in half along each direction, e.g. binary spatial partitioning. If a grid dimension is odd then the extra grid node is placed to the "right". A particular dimension (e.g. *x*, *y*, or *z*) of the grid is only split under certain conditions as well. If the aspect ratio compared to the smallest dimension for any direction is greater than or equal to two, then only those directions will be divided, thus leading to domains that have a near optimal surface area to volume ratio. If the grid is a cube of dimension 2^n then the indexing is equivalent to that of the Morton curve. The figures below illustrate how some simple domains are subdivided by this partitioning algorithm.

[2] Image from Wikipedia article on Z-order curve

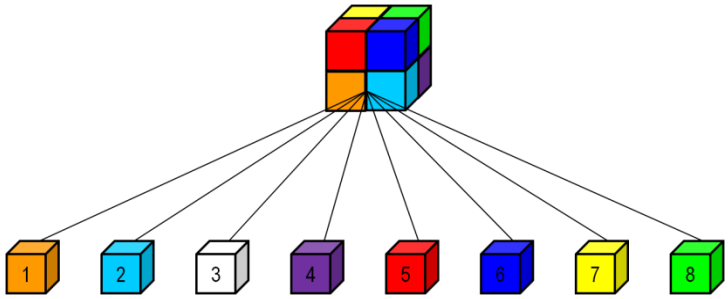


Figure 4.3 – Basic Z-Tree indexing and partitioning

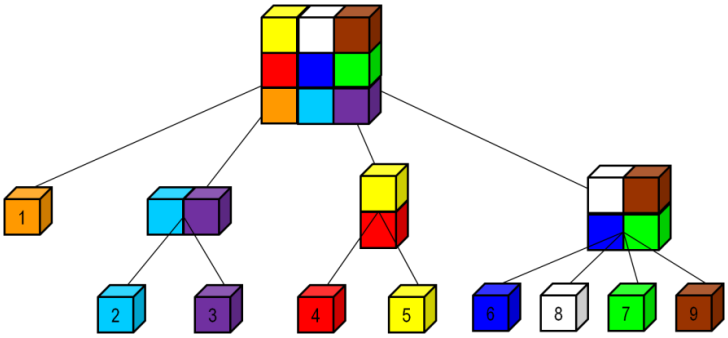


Figure 4.4 – Z-Tree partitioning for odd-sized grids

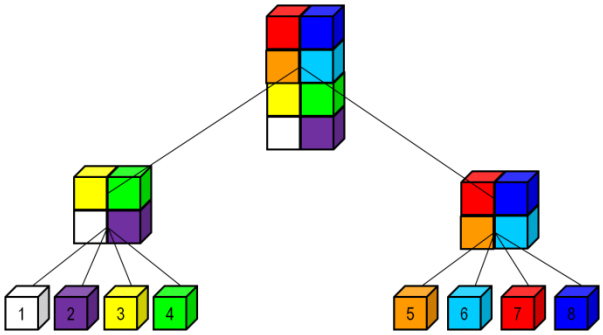


Figure 4.5 – Z-Tree partitioning for grids with high aspect ratios

For the reactor problem, the Z-Tree must also be able to handle grids with irregular outer boundaries, as depicted in Figure 4.6. This is achieved by first determining the indexing for the full rectangle (or cuboid in 3-D) and then performing a "trim" operation to remove a selected range of i, j, k grid coordinates and renumber the remaining grid points. This operation is illustrated in Figure 4.7.

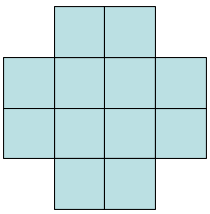


Figure 4.6 – Possible 2-D grid with irregular outer boundaries

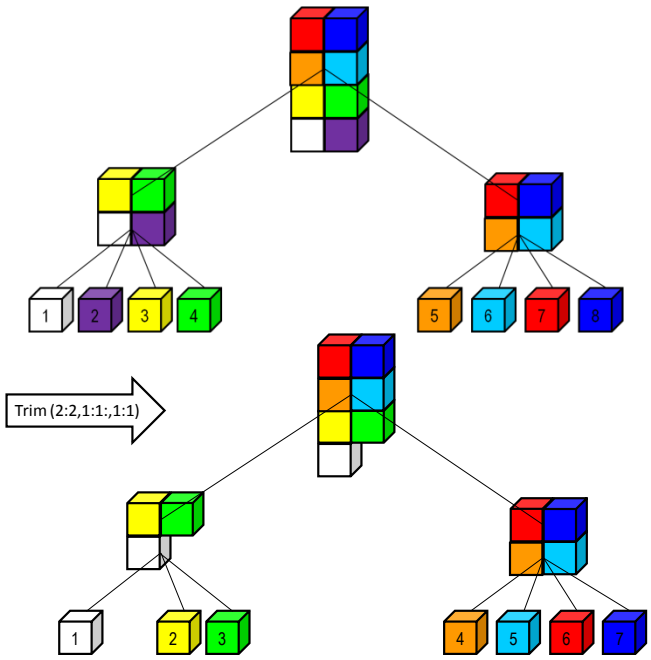


Figure 4.7 – Z-Tree trim operation

Once the grid has been converted to a Z-tree and indexed, it is a relatively straightforward to partition. Currently, the partitioning is only performed at each level of the tree to ensure that nearly equal size domains and good surface area to volume ratios are achieved. This is also done so that the partitioning ensures that there is only one neighbor per face on the subdomain. It should be noted that this is not the only way to partition the grid, and that this approach is chosen out of simplicity. Other approaches are possible and could include using different space filling curves, or requiring that the number of parallel domains be an integer factor of the total grid volume. The subdomain volumes could then be sized from the remaining prime factors of the overall grid volume after division by the number of parallel domains.

The communication of the boundary conditions is implemented using `MPI_Isend` and `MPI_Irecv`, which are non-blocking point-to-point communication operations. Individual messages are sent for all rays on a given face for a given angle and energy group. Both send and receive are initiated as soon as the sweep for a given angle has been completed. Because the communication is non-blocking the process will continue sweeping the other angles until there is no other work it can do. As will be shown in later chapters, this provides for better scaling, since most of the communication overhead can be hidden. An illustration of the timeline for execution of two processors with different subdomains sharing a boundary is shown in Figure 4.8.

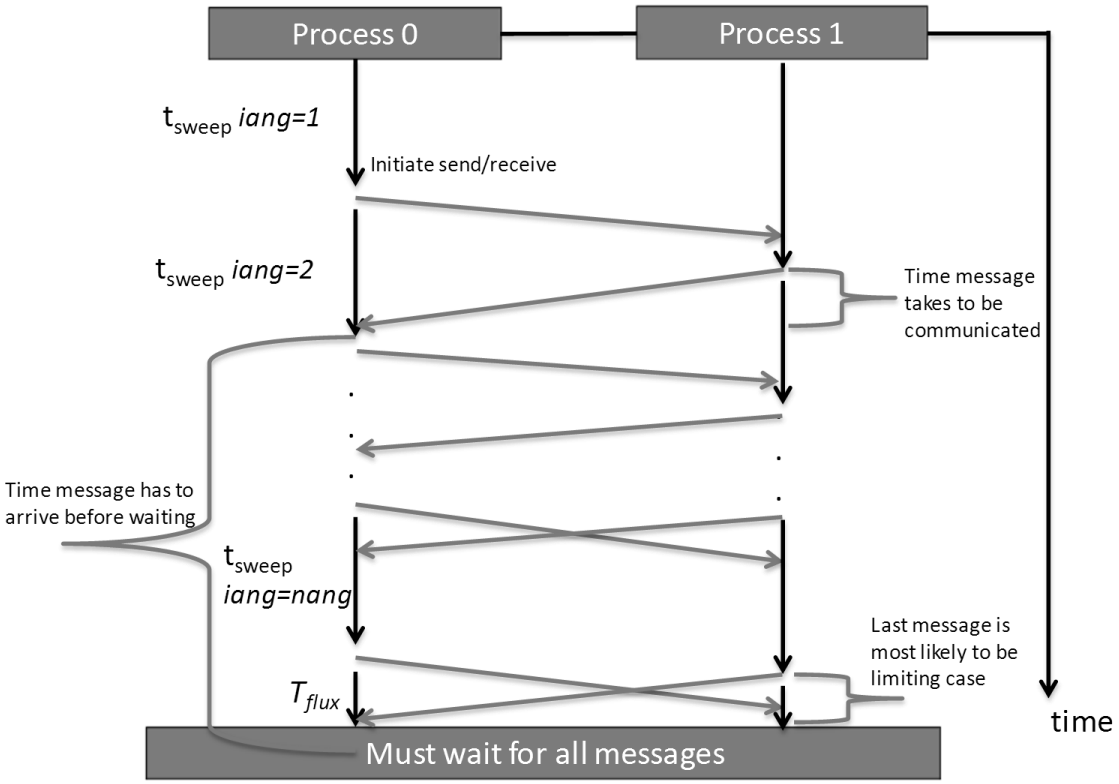


Figure 4.8 – Sequence diagram for two processes executing spatially decomposed 3-D MOC kernel in parallel

It should also be noted that other communication routines are required in the computation of the eigenvalue and convergence criteria. Specifically, a reduce operation is required in the eigenvalue calculation to compute sums or integrals over space. This is presently implemented with an `MPI_Allreduce` and is not considered in the parallel performance of the MOC kernel.

4.3 Decomposition of the Angular Domain

The angular decomposition is naturally parallel, since each direction can be swept independently. The only coupling of the solution in angle occurs within the scattering source, which requires integration over angle, and reflective boundary conditions. The angular decomposition is implemented using a distributed memory model, since this will help to alleviate the additional memory burden of storing the angular flux boundary condition incurred through the spatial decomposition.

There are generally two types of communication required of the 3-D MOC kernel for the angular decomposition. The first is a reduction operation for the computation of the scalar flux defined by Eq. (2.23). This operation is presently implemented using `MPI_Allreduce`, which requires the communication of a vector of the size of the number of discrete spatial regions in the subdomain. This operation is a collective operation and is almost always blocking. Therefore, it generally incurs more overhead, especially in the case of poor load balancing, and is known to have limited scalability. However, this should not pose a significant problem given that the expected number of angular domains is generally less than 50.

The problem posed for the partitioning of the angular domain is similar to that of the general partitioning problem within computer science. To partition the angular domain, first a weight is assigned to each discrete angle. This weight, which should be representative of the amount of work associated with this angle, is taken to be the ratio of the number of modular rays for the angle to the sum of the modular rays over all angles. Once the weights for all angles are determined, the angles are assigned to the various subdomains. The process by which the angles are assigned to each angular subdomain is based on a "greedy" algorithm, whereby the subdomain whose angles have the smallest sum of weights gets the angle with the largest weight from the set of angles still to be assigned to a subdomain. This process is also illustrated in Figure 4.9.

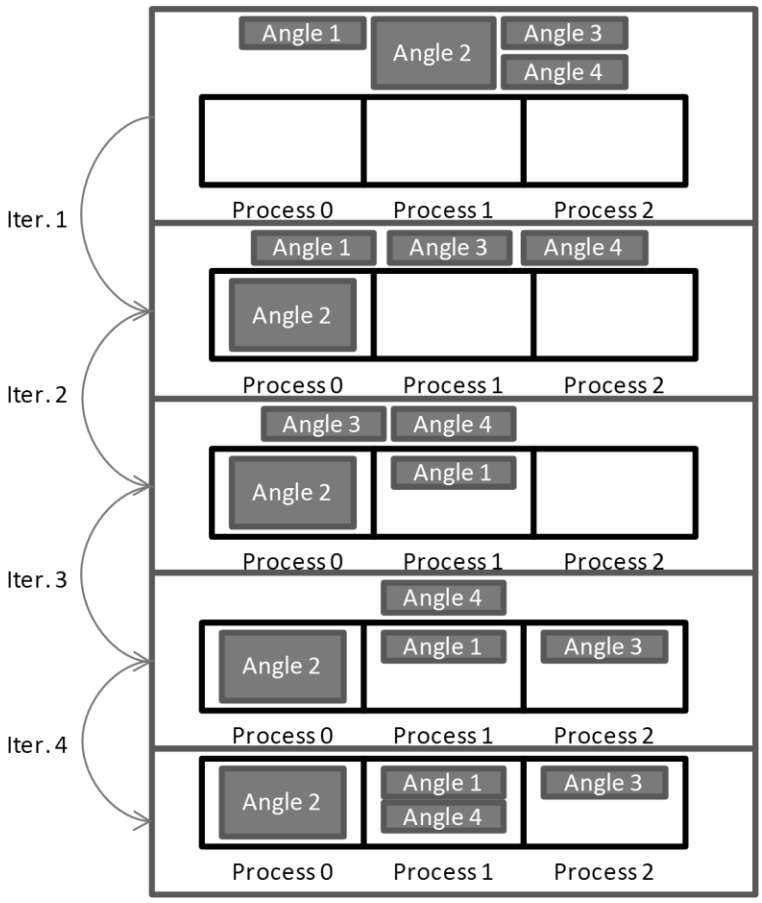


Figure 4.9 – Greedy algorithm for angular partitioning

The other type of communication overhead that may be needed for the angular decomposition is to communicate the boundary condition on a surface, specifically for the case of reflective boundary conditions. The greedy algorithm used to partition the angles does not guarantee that angles that will reflect into one another on a given surface are assigned to the same process. Thus, after the angles are assigned, one must check if there is a reflective boundary on the spatial domain. If a reflective boundary exists and the reflected angle is not on the same process as its complementary angle, then the boundary conditions will need to be communicated on this surface. The model for this communication is identical to that required by the spatial decomposition described in the previous section (4.2). Figure 4.10 shows the partitioning of the angular domain for the S_8 quadrature using 1 cm^3 ray tracing modules. The quadrature has 10 directions per octant and only half the unit sphere is stored, so there is a total of 40 discrete directions.

The figure shows the minimum relative work and maximum relative work on all domains for the given partition. It also shows the load imbalance for the partitioning and the partitioning efficiency computed as shown by Eq. (4.1) and Eq. (4.2), respectively; where $\vec{W}$ is a vector of length N_d and describes the distribution of work over the number of domains, N_d.

$$\text{Load Imbalance} \equiv 1 - \frac{\min\left(\vec{W}\right)}{\max\left(\vec{W}\right)}, \qquad \textbf{Eq. (4.1)}$$

$$\text{Partition Efficiency} \equiv \frac{N_d \times \max\left(\vec{W}\right)}{\left\|\vec{W}\right\|_1}. \qquad \textbf{Eq. (4.2)}$$

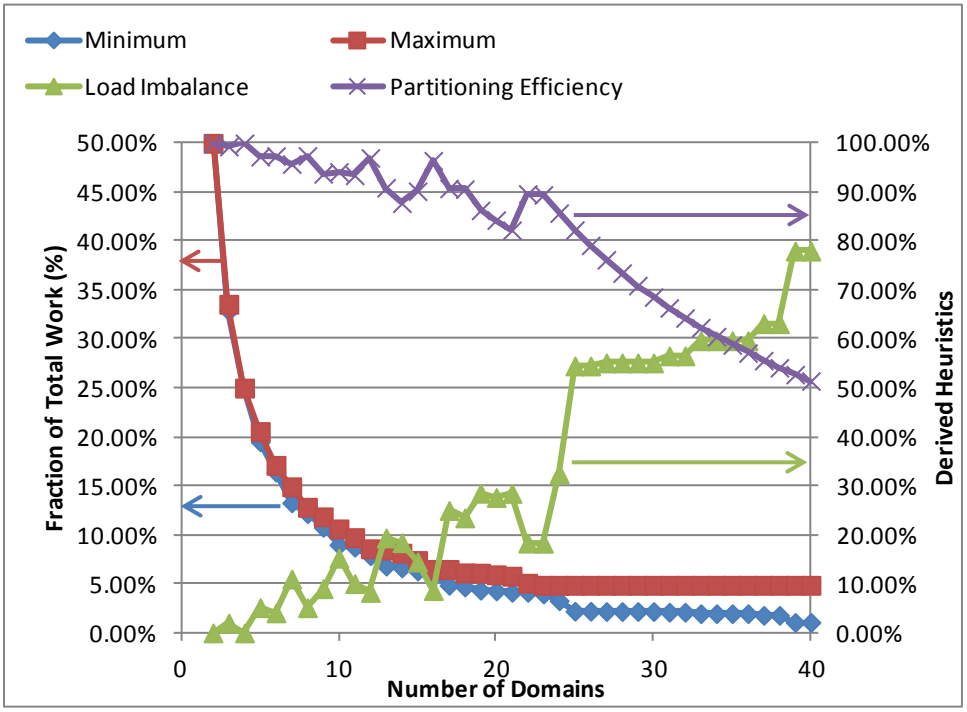


Figure 4.10 – Partitioning of rays generated from S_8 quadrature and 1 cm^3 ray tracing module

4.4 Decomposition of the Characteristics Domain

The ray decomposition involves parallelizing the loop over the long rays and the principal parallel challenge is dealing with the potential load imbalance issues since some rays are very long and others are very short, as illustrated in Figure 4.11.

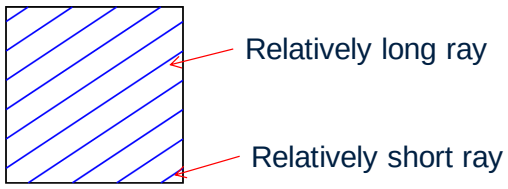


Figure 4.11 – Potential load balancing issue with rays

The ray decomposition is implemented using a shared memory model (e.g. OpenMP). For this decomposition, the main overhead comes from having to evaluate Eq. (3.12), which is now being executed by multiple threads. The approach implemented is to have each thread compute the partial sums, and after all long rays in all angles have been swept, a reduction operation is performed to sum the values accumulated on each thread to the master. For the reduction operation, a vector that is the length of the number of spatial regions, essentially storing $\hat{\varphi}_{i,g,m}$ on the right hand side of Eq. (3.13), is allocated to two dimensions with the outer dimension being the number of threads. This is essentially making the variable private to each thread, although it is shared, and avoids the need to use a critical section to avoid race conditions when evaluating Eq. (3.12) inside the parallelized loop. Consequently, the reduction over threads is moved outside the loop over angles and is performed prior to the reduction operation required by the angular decomposition.

A few additional details are related to implementation, rather than the equations that must be solved, that can be responsible for some overhead in the parallelization. The first is the overhead for creating and destroying threads when entering and exiting the threaded region of the MOC kernel. In this implementation the parallel region is defined over the whole MOC kernel to minimize the overhead for creating and destroying threads. However in doing so, this creates an issue in having to create synchronization points or single thread constructs outside of the parallelized loop. A barrier or synchronization point is required prior to updating the boundary condition (step c of Figure 3.5). This, along with the reduction operations for rays and angle and computation of the scalar flux, also only need to be performed by one thread.

The potential load balance issues are minimized by using the dynamic scheduling feature of OpenMP to distribute the rays dynamically among the threads. Although the

rays could be distributed statically onto each thread by using the number of segments in each ray as a weight, the former was chosen for ease of implementation. Figure 4.12 shows the load balancing of OpenMP's dynamic scheduler for a 2-D MOC assembly sized problem.

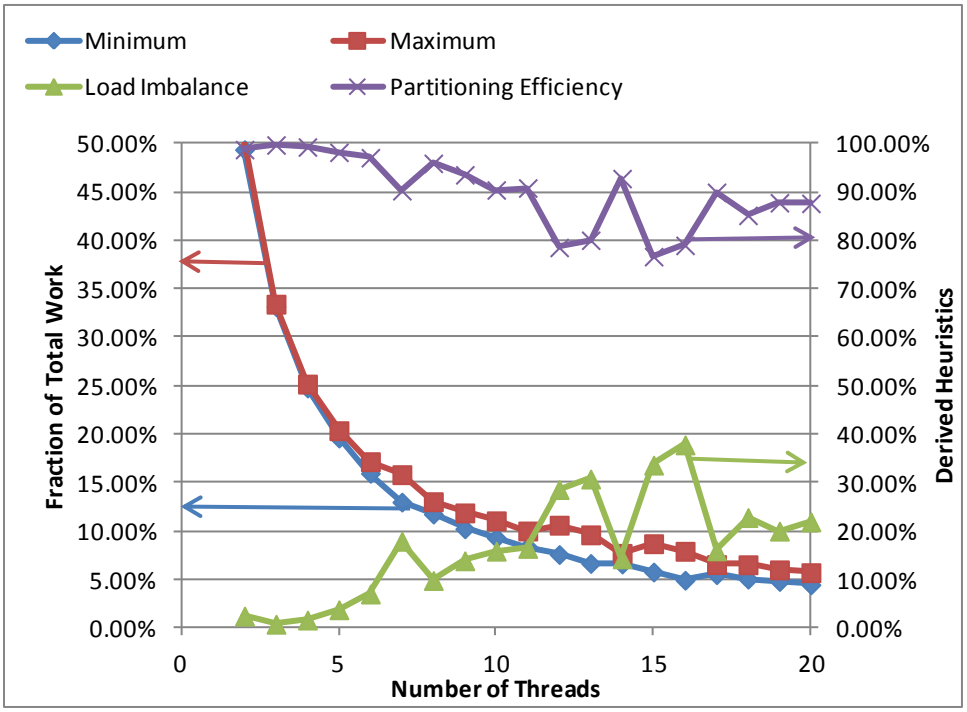


Figure 4.12 – Partitioning of Long rays with OpenMP dynamic scheduler

4.5 Summary

Figure 4.13 illustrates the method in which the whole problem is decomposed by space, angle, and ray. The problem could also be decomposed in energy, although that was not studied in this work. The serial 3-D MOC kernel described in Figure 3.5 is updated for the parallel 3-D MOC kernel with space, angle, ray decomposition and shown in Figure 4.14.

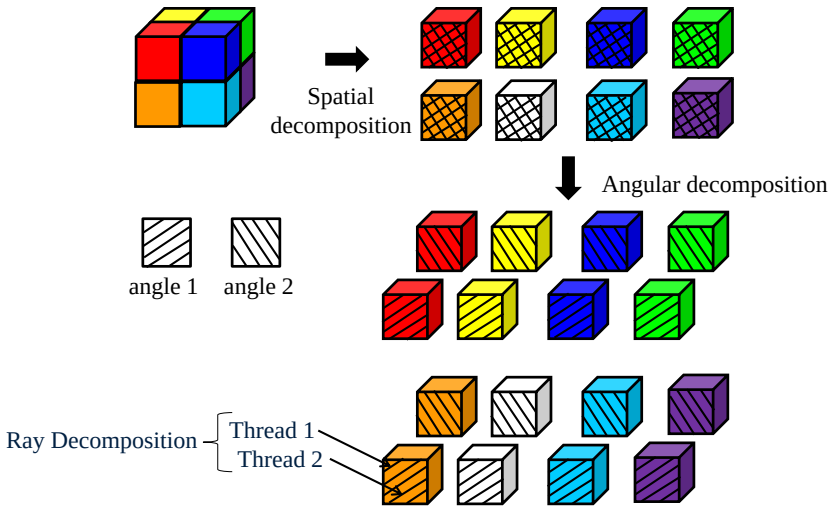


Figure 4.13 – 3-D MOC parallel decomposition scheme

1. **Loop over all angles in angle subdomain**
   - a. **Loop over all long rays in angle in parallel with threads**
     - i. Order the modular ray data for a complete long ray
     - ii. Evaluate the exponential function for all segments in the long ray
     - iii. Load incoming boundary conditions for each end of ray
     - iv. Loop over all segments in the long ray
       1. Evaluate Eq. (3.5) in forward direction
       2. Accumulate $\varphi^{d}_{i,g,m,k}$ into temporary for Eq. (3.12) for forward direction.
       3. Evaluate Eq. (3.5) in backward direction
       4. Accumulate $\varphi^{d}_{i,g,m,k}$ into temporary for Eq. (3.12) for backward direction.
     - v. Store outgoing boundary conditions for each end of ray
   - b. Accumulate $w_m \delta A_m \hat{\varphi}_{i,g,m}$ into temporary for Eq. (3.13)
   - **c. Wait for all threads to finish loop over long rays**
   - **d. Send outgoing boundary conditions to neighbor subdomains in space**
2. **Add partial sums of Eq. (3.13) from all threads**
3. **Add partial sums of Eq. (3.13) from all angular domains**
4. Evaluate Eq. (3.14) for all regions
5. **Wait for all incoming boundary conditions from neighbor subdomains in space**

Figure 4.14 – 3-D MOC kernel parallel algorithm

Chapter 5

PERFORMANCE BOUNDS MODEL DEVELOPMENT AND VALIDATION

The focus of this chapter is to detail the development of a theoretical model to predict the performance and execution time of the 3-D MOC kernel, given some basic information about problem size and hardware characteristics. This chapter first describes a conceptual model for the target computer architecture. The general equations to predict the execution time and computational performance are then developed based on this model. The 3-D MOC kernel implementation is then examined in detail, to provide upper and lower bounds on expected operation counts, which provide the coefficients in the performance model. The last section of the chapter focuses on the experimental validation of the performance model and theoretical upper and lower bounds of the kernel's operation counts.

5.1 Architecture of High Performance Compute Clusters

The target architecture assumed for this analysis is based on the common characteristics of compute clusters for high performance computing. In general, the term cluster refers to several compute nodes that are connected together via a network. This is illustrated by Figure 5.1.

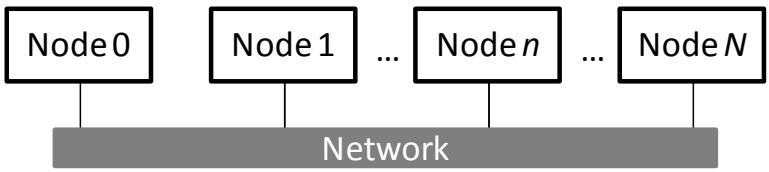


Figure 5.1 – Model cluster architecture

Frequently, clusters consist of commodity parts; for example: the CPU, motherboard, RAM and network. On a given node, the architecture may vary significantly, and may continue to evolve through exascale computing as discussed in Section 4.1. The model architecture for the node assumes only that there is some number of symmetric multi-processors (SMPs) on a node and that they share global memory. A cache memory hierarchy is also assumed on each processor. The node architecture is illustrated in Figure 5.2.

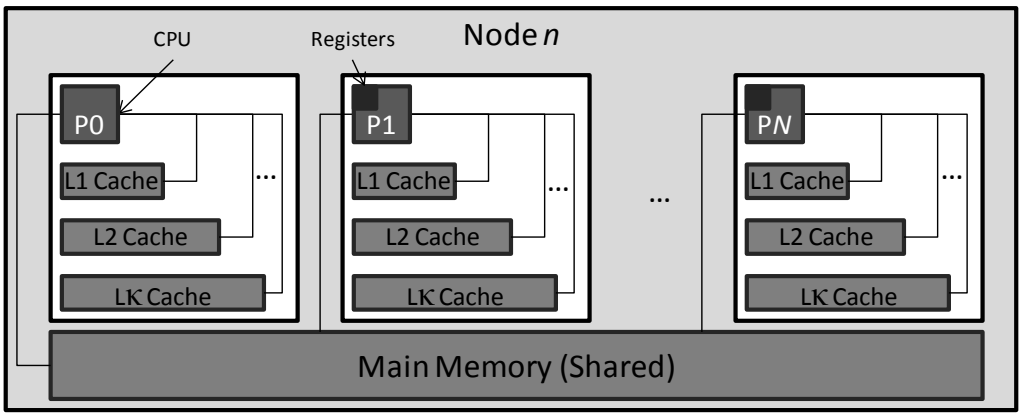


Figure 5.2 – Model node architecture

This model architecture assumes nothing about how the different nodes are physically connected or the underlying hardware used to connect them. Furthermore, the model architecture does not specify certain details of the node architecture, such as the number of floating point units or processors, number of levels of cache or how processors might share these resources. In Table 5.1 the hardware properties that are relevant to the performance model of Section 5.2 are defined.

Table 5.1 – Model architecture hardware performance properties

Symbol	Name	Example Units	Meaning
C	Clock Speed	cycles/s	Number of processor clock cycles per unit time
t_f	time per FLOP	cycles/FLOP	Number of processor clock cycles required to execute a floating point instruction
α_j	Cache Latency	ns/access	Time required to load data onto the processor for the j^{th} level of cache
l_j	Cache Line Size	bytes	Largest amount of memory that can be loaded into cache at once for the j^{th} level of cache
α_{mem}	Memory Latency	cycles/access	Time required to load data onto the processor from main memory
$\alpha_{network}$	Network Latency	μs	Time required to transmit a single packet from one node to another
$\beta_{network}$	Inverse Network Bandwidth	s/MB	The inverse of the largest message size that may be transmitted in a single packet over the network.

5.2 Basic Equations of a Performance Model

For the performance of an algorithm in scientific computing the conventional metric is the number of *floating point operations* (FLOPs) *per unit time*; typically this is expressed in units of millions of floating point operations per second or MFLOPS[3]. The basic equations presented here come from [49]. Eq. (5.1) is the equation that will be used for performance.

$$P \equiv \frac{F}{T}. \qquad \textbf{Eq. (5.1)}$$

Here F is the total number of FLOPs executed and T is the execution time. It is also often valuable to compare measured performance against the hardware's theoretical peak performance to obtain a fraction of the theoretical peak. The theoretical peak performance is defined in terms of the hardware properties of Table 5.1 as:

$$P_{peak} = C/t_f \ . \qquad \textbf{Eq. (5.2)}$$

In Eq. (5.1) F and T are naturally going to be functions of the problem size. These functions are derived in terms of the problem size factors for the 3-D MOC kernel detailed in Section 5.3. In general, F is a measure of the number of a given operation, and

[3] FLOPs, with a "s", denotes plural of FLOP (floating point operation) while FLOPS, with a "S", denotes FLOPs per second.

therefore is solely a function of the algorithm. However, the function for execution time is also a function of the hardware properties. A latency-based model for the execution time is shown in Eq. (5.3) where the memory access times to each level of the memory hierarchy are treated explicitly.

$$T = Ft_F + \alpha_1 L + \sum_{j=1}^{\kappa-1}\left(\alpha_{j+1} - \alpha_j\right)M_j + \left(\alpha_{mem} - \alpha_\kappa\right)M_\kappa. \qquad \textbf{Eq. (5.3)}$$

In this equation, α is the memory access time or cache latency, M is the number of cache misses at a given level of cache, and κ is the number of levels of cache on the machine. L is the number of load operations; where a load operation consists of moving a piece of data, such as a word or byte, from memory to a register on the processor. A cache miss occurs when a processor attempts to load some data from a given cache level and the data is not present in that cache level,. Therefore the processor must retrieve this data from a higher level of cache or main memory resulting in a cache miss.

As mentioned previously, F is a function of the algorithm. The other terms in Eq. (5.3) that are not strictly a function of the hardware are L and M. L is similar to F in that it is solely a function of the algorithm. However, the number of cache misses, M is somewhat more complicated since, it is strongly influenced by both the algorithm and the hardware and typically cannot be known exactly. Therefore, the performance of the 3-D MOC kernel will be bounded using reasonable assumptions to provide upper and lower bounds for the cache misses. The focus of the next section is to analyze the algorithm implemented for the 3-D MOC kernel described in Chapter 3. The objective is to determine F, L, and the upper and lower bounds for M; all in terms of parameters that describe the problem size for the 3-D MOC kernel.

5.3 Performance Bounds of 3-D MOC Kernel

As noted in Section 3.3, the 3-D MOC kernel is quite complex compared to other conventional kernels in scientific computing (e.g. matrix-matrix multiplication). Therefore, it is advantageous to think of the 3-D MOC kernel as a collection of micro-kernels. These micro-kernels are listed in Table 3.2. The following section proceeds with this taxonomy and develops the detailed expressions for F and L for each component

micro-kernel as a function of the problem size. Then Section 5.3.2 lists the assumptions used to bound *M* and develops the expressions for these upper and lower bounds. Sections 5.3.1 and 5.3.2 only consider serial execution, so in Section 5.3.3 the list of components and expressions for *F*, *L*, and *M* are extended for the parallel model; adding terms for the number of parallel domains for each decomposition discussed in Chapter 4.

5.3.1 Component-Based Description of Kernel

From the list of micro-kernels in the Table 3.2, the execution time for a single transport sweep of the 3-D MOC kernel is expanded as:

$$T_{sweep} = T_{build} + T_{\exp} + T_{MOC} + T_{BC} + T_{scal} + T_{BCUp} + T_{flux}. \qquad \textbf{Eq. (5.4)}$$

Similarly, for the expression for the number of FLOPs is expanded as:

$$F_{sweep} = F_{build} + F_{\exp} + F_{MOC} + F_{BC} + F_{scal} + F_{BCUp} + F_{flux}. \qquad \textbf{Eq. (5.5)}$$

The FLOP count for each component can be determined by examining equations Eq. (3.3), Eq. (3.12), Eq. (3.13), and Eq. (3.14). In the *build* component the segment lengths are multiplied with the total neutron cross section. This operation is performed exactly once for every discrete segment in the problem, so F_{build} = nseg. Next in the *exp* component, a linear interpolation of tabulated values is performed. To compute the table index and then perform the linear interpolation requires 3 FLOPs. The exponential is then evaluated for every discrete segment in the problem exactly once, and therefore $F_{exp} = 3 \times \text{nseg}$. In the MOC component Eq. (3.3) and Eq. (3.12) are evaluated, and since the exponential term has already been evaluated, the number of FLOPs to evaluate Eq. (3.3) for each segment is 3 and for Eq. (3.12) it is just one FLOP. In Figure 3.5 these equations are evaluated twice for each segment, for the forward and backward direction along the characteristic ray, and therefore $F_{MOC} = 8 \times \text{nseg}$.

The *BC* and *BCUp* components only move data in memory and do not perform any FLOPs, so F_{BC} = 0 and F_{BCUp} = 0. Eq. (3.13) scales the partial sums of Eq. (3.12) with the angular weights, which requires 2 FLOPs for each discrete spatial region and for all discrete angles. Because $\text{nang} = 4 \times \text{nangoct}$, $F_{scal} = 8 \times \text{nreg} \times \text{nangoct}$. Finally,

evaluating the scalar flux as shown by Eq. (3.14) requires 4 FLOPs for each region, and therefore $F_{flux} = 4 \times \text{nreg}$.

To evaluate each component of Eq. (5.4) the number of loads in each component must be known as required by Eq. (5.3). L_{build} is discussed last, as it will become evident that the memory access patterns here are difficult to generalize for all problems. Starting with L_{exp}, the evaluation of the table first requires that the table index be determined, which requires 2 loads. The segment optical thickness must then be determined which requires accessing two coefficients in the exponential table and a location to store the evaluated exponential. Again, the table is evaluated once for all segments, so $L_{exp} = 6 \times \text{nseg}$.

For L_{BC}, the boundary condition for each end of each ray are copied from the global data structures to local arrays and then back. Each copy requires 2 loads, so $L_{BC} = 8 \times \text{nlongray}$. The boundary condition update, L_{BCUp}, requires copying the outgoing boundary conditions to the incoming boundary conditions for each end of each long ray, therefore $L_{BCUp} = 4 \times \text{nlongray}$.

In the solution of the MOC equations, to evaluate a single direction requires loading the global region index, the incoming and outgoing angular flux, region source, the evaluated exponential term, and performing the partial sum of Eq. (3.12). This results in 6 loads, so when evaluating both directions, which is done for all segments, $L_{MOC} = 12 \times \text{nseg}$. When scaling the angular flux for each angle, it requires two loads for each region, and one load up front for the weight, so $L_{scal} = 8 \times \text{nreg} \times \text{nangoct} + 4 \times \text{nangoct}$. Next, for the computation of the scalar flux by Eq. (3.14), the region-wise source, volume, and total cross section must be loaded along with the scalar flux. In the implementation of this equation the same memory locations are used to store $\phi_{i,g}$ and $\hat{\phi}_{i,g}$, so a fifth load is not needed. Therefore, $L_{flux} = 4 \times \text{nreg}$.

In order to quantify L_{build}, it is necessary to locate the starting surface and module of the ray. There is then a loop over the modular rays within this long ray, and inside of this loop there is another loop over segments in the modular ray. Ultimately, a loop is performed over each segment in the problem. However, it is not possible to know *a priori*

how many loads and stores are required in order to move through the modular ray data. The number of loads in this part is strongly dependent on the modularity of the problem, e.g. the number of modular rays per long ray and the number of segments per modular ray. Consequently, the best estimate of the number of loads is going to be a problem dependent constant, c_{build}, multiplied by the number of segments. In other words, it is postulated that the number of loads in the *build* component is proportional to the number of segments. It may also be observed that at as a minimum the region index, the total cross section, the segment length, and the optical thickness are loaded for each segment. Therefore, it is expected that $c_{build} > 4 \times \text{nseg}$.

The number of FLOPs and loads for each component are summarized in Table 5.2. The ratio of the FLOPs to loads is also shown, which can be used as a metric for the computational intensity of the algorithm. The computational intensity is a good indicator of the maximum achievable performance of the algorithm, which is independent of any computer architecture.

Table 5.2 – Number of FLOPs and Loads for serial 3-D MOC kernel

Component	FLOPs	Loads	Computational Intensity (FLOPs/Loads)
build	nseg	$c_{build} \times \text{nseg}$	$1/c_{build}$
exp	$3 \times \text{nseg}$	$6 \times \text{nseg}$	0.5
MOC	$8 \times \text{nseg}$	$12 \times \text{nseg}$	0.75
BC	0	$8 \times \text{nlongray}$	0
BCUp	0	$4 \times \text{nlongray}$	0
scal	$8 \times \text{nreg} \times \text{nangoct}$	$8 \times \text{nreg} \times \text{nangoct} + 4 \times \text{nangoct}$	~1.0
flux	$4 \times \text{nreg}$	$4 \times \text{nreg}$	1.0
Sweep (Total)	$12 \times \text{nseg} + 8 \times \text{nangoct} \times \text{nreg} + 4 \times \text{nreg}$	$(18 + c_{build}) \times \text{nseg} + 12 \times \text{nlongray} + 8 \times \text{nangoct} \times \text{nreg} + 4 \times \text{nreg} + 4 \times \text{nangoct}$	$0.0 < C.I. <\sim 0.5$

5.3.2 Cache Miss Bounds

In bounding the cache misses for the MOC kernel, several guiding assumptions are required. First, the conflict misses and capacity misses are ignored, and therefore cache misses refers only to compulsory misses. This assumption is basically equivalent to assuming an infinite cache capacity and fully associative caches.

For each component the lower bound assumes that there are no cache misses. While this will never be observed in practice, except for perhaps trivially small problems, it nonetheless provides a useful assumption for the lower bound. Assuming the lower bound of cache misses is 0 greatly simplifies Eq. (5.3), and subsequently provides the absolute lower bound for the execution time, which will become solely a function of the computational intensity shown in Table 5.2. The analysis in Chapter 6 and Section 5.4.6 of this chapter make use of this assumption to provide insight about the overall efficiency expected from MOC kernels regardless of the computer architecture, which is valuable when comparing the MOC method to other transport methods.

For the upper bound on cache misses, the basic assumption made is that all the data operands needed for computation within a given component reside entirely in global memory and not in any level of the cache. Again, this is an assumption that will typically not be true in practice, since many of the components reuse the same data operands. However, this is still a useful assumption to make because it provides a true upper bound in assuming the slowest memory access time for all memory accesses.

For each component in the kernel the lower and upper bounds of the cache misses for the j^{th} level of cache are computed using Eq. (5.6) and Eq. (5.7), respectively:

$$M_{lower,j} = 0, \qquad \textbf{Eq. (5.6)}$$

$$M_{upper,j} = \frac{L}{l_j}. \qquad \textbf{Eq. (5.7)}$$

5.3.3 Parallel Overhead

This section focuses on the additional operations that are added to the 3-D MOC kernel because of the parallel decompositions discussed in Chapter 4. In an efficient spatial decomposition, volume-dependent quantities such as `nseg` and `nreg` are evenly divided over the domains, so long as the modular domains are equivalent. However, the number of long rays reduces linearly with the reduction in surface area of the subdomain. Consequently, the execution times of each component are reduced by roughly the same factor. However, if there is a poor load balance, then execution time becomes a function of the max of the problem size values over all spatial domains.

For the overhead of the spatial decomposition, the time to update the boundary condition, T_{BCUp}, will be reduced by a factor of $(p_{space})^{2/3}$, which is related to the reduction in surface area to volume when a cuboid region is subdivided. It will also include an additional factor for the time to communicate the boundary conditions via the spatial decomposition communication scheme. This communication scheme uses point-to-point communication. In general, the model for the execution time of point-to-point communication is given as a function of the message size, N, and the latency and inverse of the bandwidth of the network used to perform the communication. The equation for the time to perform point-to-point communication is shown in Eq. (5.8):

$$T_{comm} = \alpha_{network} + \beta_{network} N. \qquad \textbf{Eq. (5.8)}$$

As noted in Figure 4.8, the communication of the boundary conditions occurs with multiple messages. A message is sent for each face and each angle, so the message size will be a function of the number of long rays of a given angle intersecting a given face which may be written as:

$$N_{space} = \text{nlongray}(iang, iface). \qquad \textbf{Eq. (5.9)}$$

Typically, the message sizes for different angles and faces will vary considerably, so it is difficult to develop *a priori* an exact expression for Eq. (5.9). However, it is trivial to determine this information at run time for a given problem.

Also, one may note that in Figure 4.8 the communication is non-blocking, meaning it is overlapped with useful work, and that the last message sent will have the least amount of work remaining to hide the communication time. This leads to the following modification of Eq. (5.4) to Eq. (5.10) to account for the execution time with spatial decomposition. In Eq. (5.10) p_{space} is the number of decomposed spatial domains being executed in parallel, and T_{space} replaces T_{BCUp} to account for the time to update the boundary condition. Because the overhead of the spatial decomposition is only the communication of data and no additional FLOPs are required compared to serial execution, Eq. (5.5) is not modified except to account for the number of processors used for the spatial decomposition and to remove F_{BCUp} since this is zero.

$$T_{sweep} = \frac{T_{build} + T_{\exp} + T_{MOC} + T_{BC} + T_{scal}}{p_{space}} + \max\left(\frac{T_{flux}}{p_{space}}, T_{space} \right), \qquad \textbf{Eq. (5.10)}$$

$$F_{sweep} = \frac{F_{build} + F_{\exp} + F_{MOC} + F_{BC} + F_{scal} + F_{flux}}{p_{space}}, \qquad \textbf{Eq. (5.11)}$$

$$T_{space} = \text{nface} \times \left(\alpha_{network} + \beta_{network} \times N_{space} \right) + \frac{T_{BCUp}}{\left(p_{space} \right)^{2/3}}. \qquad \textbf{Eq. (5.12)}$$

For the angular decomposition, only those parameters of problem size directly related to angle are reduced. This includes nseg and nlongray, but not nreg. The overhead from the angular decomposition, as noted in Chapter 4, is an all reduce operation. Since this is operation is implemented using `MPI_Allreduce` its execution time may vary with MPI library. Nonetheless, for those algorithms found in the open literature [50], [51] the time to perform this operation is reported as either one of Eq. (5.13) or Eq. (5.14), depending on the message size. Since the message size for this operation in the 3-D MOC kernel is `nreg`, the algorithm for large message sizes is assumed. This operation also requires additional FLOPs, which are also performed in parallel, represented by γ which is the computation cost per unit memory.

$$T_{\text{All\_reduce,small}} = \lceil \log p \rceil (\alpha_{network} + \beta_{network} \times N + \gamma \times N), \qquad \textbf{Eq. (5.13)}$$

$$T_{\text{All\_reduce,large}} = 2 \log p \alpha_{network} + \frac{p-1}{p}(2\beta_{network} \times N + \gamma \times N). \qquad \textbf{Eq. (5.14)}$$

In the angular decomposition the message size is equal the number of regions on a spatial subdomain. Eq. (5.10) is then modified as follows to account for the angular decomposition, p_{ang}, which represents the number of processors used for the angular domain. Once again, Eq. (5.5) for the FLOPs of the MOC kernel is not modified except to account for the number of processes used in the angle decomposition since the additional FLOPs of the reduction operation are included in Eq. (5.14).

$$T_{sweep} = \frac{T_{build} + T_{\exp} + T_{MOC} + T_{BC} + T_{scal}}{p_{space} p_{ang}} + \max\left(\frac{T_{flux}}{p_{space}} + T_{ang}, T_{space} \right), \qquad \textbf{Eq. (5.15)}$$

$$F_{sweep} = \frac{F_{build} + F_{\exp} + F_{MOC} + F_{BC} + F_{scal}}{p_{space} p_{ang}} + \frac{F_{flux}}{p_{space}}, \qquad \textbf{Eq. (5.16)}$$

$$T_{\text{ang}} = 2 \log p_{ang} \alpha_{network} + \frac{p_{ang} - 1}{p_{ang}}(2\beta_{network} + \gamma) \times \frac{\text{nreg}}{p_{space}}. \qquad \textbf{Eq. (5.17)}$$

Finally, for the ray decomposition only the loop over long rays is parallelized and the added overhead is another reduction operation in order to reduce the partial sums across threads onto the master thread. The overhead for the reduction over threads adds $p_{ray} \times \text{nreg}$ FLOPS per sweep. The evaluation of the execution time, T_{ray}, is also based on Eq. (5.3), and the additional number of loads, L_{ray}, for this component is $2 \times p_{ray} \times \text{nreg}$. Furthermore, the OpenMP routines called at run time have some overhead. These routines are typically compiler or machine dependent, and their overhead is represented by introducing a new term, T_{OMP}, defined in Eq. (5.20), where the subscript refers to the OpenMP directive or construct, and "chunk" refers the chunk size

used for the scheduling directive. One may note that, in general, this equation should be consistent with the implementation, and there are multiple ways to implement the OpenMP parallelism; particularly with respect to the choice of loop scheduling and thread synchronization, which can have huge affects on performance. Furthermore, it is likely the case that little is known about the supplied OpenMP run time library, consequently the various factors in Eq. (5.20) should be determined experimentally for a given platform and library. Thus, after introducing the ray parallelism into Eq. (5.15) and Eq. (5.16) the final form of the performance model becomes:

$$T_{sweep} = \frac{T_{build} + T_{\exp} + T_{MOC} + T_{BC}}{p_{space} p_{ang} p_{ray}} + \frac{T_{scal}}{p_{space} p_{ang}} + T_{OMP}\left(p_{ray}\right) + \max\left(\frac{T_{flux} + T_{ray}}{p_{space}} + T_{ang}, T_{space}\right), \quad \textbf{Eq. (5.18)}$$

$$F_{sweep} = \frac{F_{build} + F_{\exp} + F_{MOC} + F_{BC}}{p_{space} p_{ang} p_{ray}} + \frac{F_{scal}}{p_{space} p_{ang}} + \frac{F_{flux} + F_{ray}}{p_{space}}, \quad \textbf{Eq. (5.19)}$$

$$T_{OMP}\left(p_{ray}\right) = T_{PARALLEL}\left(p_{ray}\right) + \left(2T_{BARRIER}\left(p_{ray}\right) + T_{SINGLE}\left(p_{ray}\right)\right) \times 4\text{nangoct} + \left\lceil \frac{\text{nlongray}}{p_{ray} \times \text{chunk}} \right\rceil T_{SCHEDULE}\left(p_{ray}\right). \quad \textbf{Eq. (5.20)}$$

Eq. (5.18) and Eq. (5.19) provide a theoretical basis for predicting the performance of the parallel 3-D MOC kernel described in Figure 4.14. However, before this model can be used with any confidence, it should be validated against measurement to ensure that the kernel description and implementation are consistent with the models and operation counts described in this section and Section 5.2.

5.4 Experimental Evaluation of Performance Model

The source code was *instrumented* and *profiled* in order to evaluate the performance model developed thus far in this chapter. Instrumenting refers to the process of modifying the source code to take a measurement, and profiling refers to the process of collecting

time integrated quantities from typical executions of the code. As with any experimental measurement, there are several best practices when designing a good experiment and collecting accurate measurement data. Some important considerations include understanding the effects of instrumentation and the test environment. When a section of code is instrumented, it affects how the compiler converts the source code into executable code. This can have a considerable impact on the effectiveness of the compiler optimizations. Furthermore, the instrumentation code that must be added to collect the measurement data requires the use of resources in the computer, and therefore the instrumentation incurs its own overhead during execution. If the test machine being used to run the instrumented executable is a shared resource, then machine load is also an important factor that must be considered in analyzing measurement results. The measurement of some quantities is also far easier than others, as will become apparent later in the section. Measuring the number of FLOPs can be done with no uncertainty, but measuring L1 cache accesses and cache misses can have considerable uncertainty because of compiler optimizations and the overhead introduced by the measurement itself. In summary, to properly understand measurement data, it is essential to understand the above effects.

The remainder of this section presents the system used to collect performance data, the techniques by which the hardware coefficients listed in Table 5.1 are determined, and a description of the test problems used to collect performance data. The results from the performance experiments are then analyzed to show that the models of Section 5.3 provide reasonable accuracy in predicting the performance of the 3-D MOC algorithm.

5.4.1 Measurement system

The test machine for the performance model evaluation was sunspear.engin.umich.edu, a Linux workstation running Ubuntu 12.0.4. Sunspear's architecture includes a 4 socket motherboard, each having an AMD Opteron™ 6238. The AMD Opteron™ 6238 processor has 12 cores and 6 floating point units. The architecture of a single socket on the Sunspear is illustrated in Figure 5.3.

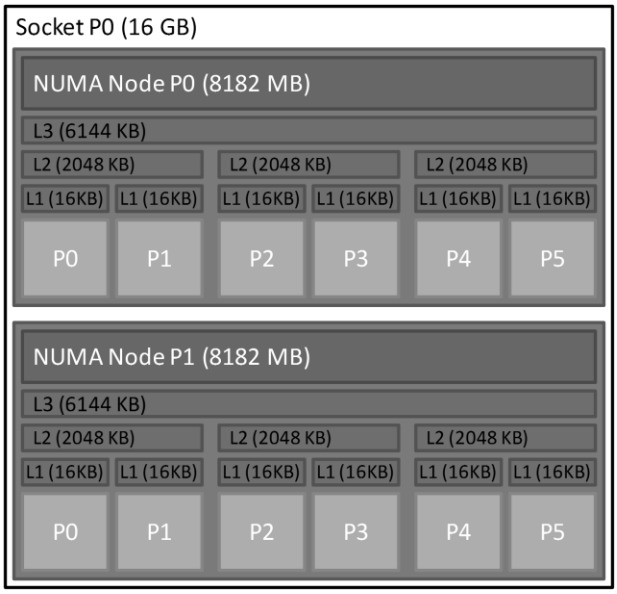


Figure 5.3 – Socket architecture of AMD Opteron™ 6238 on Sunspear

The measurements were obtained with only other minimal OS processes running concurrently, and therefore the machine had no abnormal or varying loads. For the base components of the kernel represented in Eq. (5.4) and Eq. (5.5) the measurements were performed using a single core.

In all cases, the executables were generated with the GNU 4.6.3 compiler with OpenMP and the OpenMPI 1.6 implementation of MPI. Some experiments were run without optimization using the `-O0` compiler flag, and others were run with optimizations using the `-Ofast` compiler flag. The results indicate which compilation was used to generate them. More advanced optimizations were avoided that take advantage of advanced machine specific instruction sets, such as Streaming SIMD Extensions (SSE) or Advanced Vector Extensions (AVX). Additionally, when measuring the components of base 3-D MOC kernel, a separate executable was generated for measuring each component so as to restrict the instrumentation to only the component of interest and thereby minimize the instrumentation overhead. In the measurements obtained, the data analysis includes an adjustment for the overhead from instrumentation, which was estimated using the code illustrated by Figure 5.4.

```
n=0
DO WHILE (elapsed_time < one_second)
  n=n+1
  tstart=tic()
  DO i=1,10000
    CALL BEGIN_MEASUREMENT()
    CALL END_MEASUREMENT()
  ENDDO
  tstop=toc()
  elapsed_time=tstop-tstart
ENDDO

!Average measurements over n*10000 samples
```

Figure 5.4 – Pseudo-code for estimating instrumentation overhead

The measurements were obtained using the Performance Application Programming Interface (PAPI) library v4.4.0.0 [52]. In addition to other features, the PAPI interface provides subroutine interfaces in Fortran for timing a section of code with nanosecond precision, and also for counting certain hardware events. The PAPI library interfaces with a machine's OS kernel to read data from processor hardware counters. Many modern processors include a dedicated set of registers on the chip for counting specified events within the hardware. The PAPI library provides several preset hardware events from which a developer can choose to count. They are each given a standardized name by the PAPI library. The hardware events that were measured in this study are listed in Table 5.3, which has the PAPI preset event name and description.

Table 5.3 – List of measured hardware events

PAPI Library Preset Event Name	Description
PAPI_FP_OPS	Counts floating point operations
PAPI_L1_DCA	Counts L1 data cache accesses
PAPI_L1_DCM	Counts L1 data cache misses
PAPI_L2_DCM	Counts L2 data cache misses

It should be noted that the test machine has three levels of cache, although it is not possible with the PAPI library to measure cache misses at this level. This somewhat

complicates the following analysis, but it appears from the results that there is only a minimal contribution to execution time from the L3 cache misses.

5.4.2 Determining Hardware Coefficients

The hardware coefficients listed in Table 5.1 must be determined to evaluate the execution time model of Eq. (5.3). Table 5.4 below shows the methods by which each of these values was determined for Sunspear. In general, these procedures may be repeated on almost any platform to obtain the necessary values required by the performance model presented in the previous sections. Much of the data can be obtained from the machine or vendor documentation. However, cache access latencies are typically not reported, and therefore micro-benchmarks were used as an alternative to determine these values as well as a few others. It is also important to note that some of these values, especially the cache access latencies, may encompass a range of values. This is due to the inherent difficulties in measuring such quantities, and also due to limitations in the performance model which does not necessarily account for all of the hardware on the computer involved in the given operation.

Table 5.4 – Methods used to obtain values of performance model hardware coefficients for Sunspear

Symbol	Name	Method used to obtain value
C	Clock Speed	Machine/vendor documentation
t_f	time per FLOP	Computed from machine/vendor documentation with some assumptions
α_j	Cache Latency	Deduced from the Saavedra-Barrera micro-benchmark
l_j	Cache Line Size	Machine/vendor documentation
α_{mem}	Memory Latency	Deduced from the Saavedra-Barrera micro-benchmark
$\alpha_{network}$	Network Latency	OMB MPI Tests from NERSC Trinity Benchmarks
$\beta_{network}$	Inverse Network Bandwidth	OMB MPI Tests from NERSC Trinity Benchmarks

The clock speed of AMD Opteron™ 6238 is reported as 2.6GHz and the cache line size of the machine is 64 bytes for all three levels of cache. The floating point unit has two 128-bit floating point multiply accumulators (FMACs). and therefore it can complete four 64-bit floating point operations in each cycle. However, since this test does not use advanced instruction sets like SSE or AVX it is more likely that each FMAC can only execute 2 floating point operations per cycle. Typically it is very difficult to achieve the peak FLOPs/cycle during execution unless some detailed information about the

specific compiler and target processor is known. Therefore, t_f is given a range of values to account for the uncertainty of how efficiently the compiler may be able to convert source code to the processor's instruction set.

The cache access latencies are determined using the Saavedra-Barrera micro-benchmark [53]. The basic purpose of this benchmark is to measure how long it takes to loop through an array using different strides, and then to repeat this process for arrays of different sizes. The resulting output of this benchmark when run on Sunspear is shown in Figure 5.5.

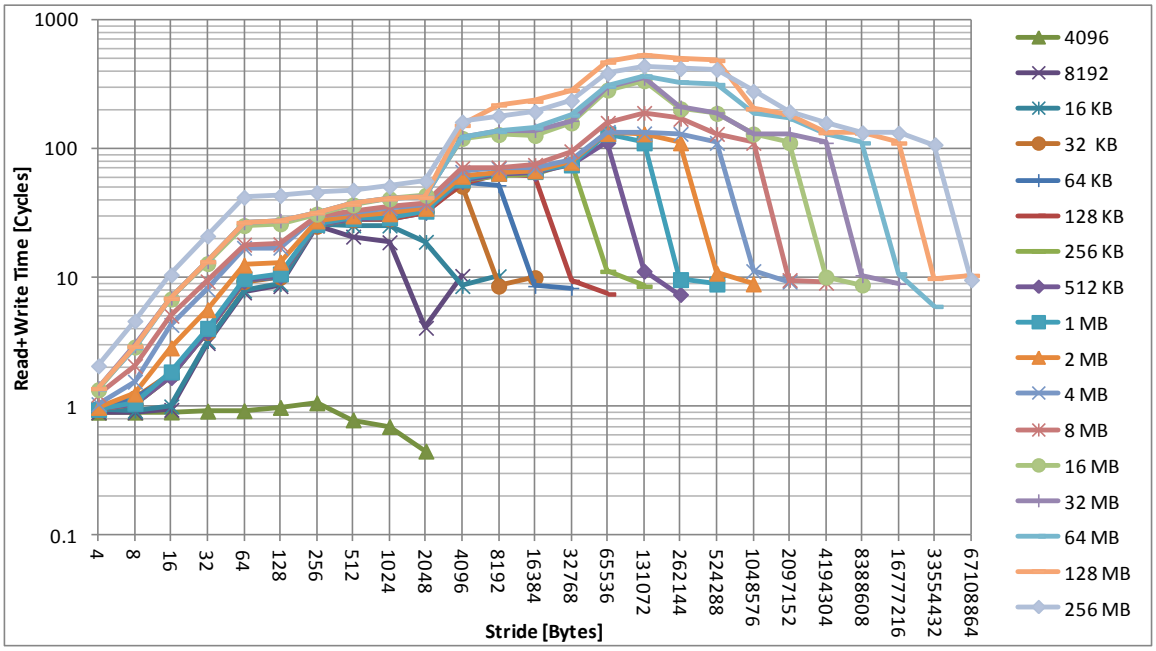


Figure 5.5 – Saavedra-Barrera benchmark results

It is apparent in these results that there are several plateaus that correspond to the different access times to the different levels of cache. In some cases there is a range, which can result from varying loads on the hardware or be attributed to hardware components at work that are not assumed to be present in the model. One other important observation about Figure 5.5 is the 4096 B case shows sub-cycle performance. This is not physically possible but is observed due to insufficient precision in the timing measurements of the benchmark. The first plateau is ~1 cycle, which corresponds to the L1 cache as this is the access latency observed for most of the unit stride cases. The next plateau is ~8-10 cycles and corresponds with the L2 cache. Beyond this the plateaus are

more difficult to discern, but the L3 cache access time is likely between ~20-110 cycles. The access time to main memory should be the highest plateau, and for the array sizes that were tested this upper asymptote was not reached. The largest arrays had their peak access times between ~300-500 cycles, so memory access time is at least 300 cycles, but the upper range is not known exactly. The final list of hardware values for Sunspear are shown in Table 5.5.

Table 5.5 – Performance model hardware values for Sunspear

Symbol	Name	Value
C	Clock Speed	2.6 GHz
t_f	time per FLOP	0.25-1 FLOPs/cycles
α_1	L1 Cache Latency	~1 cycle
α_2	L2 Cache Latency	~8-10 cycles
α_3	L3 Cache Latency	~20-110 cycles
l_1	L1 Cache Line Size	64 bytes
l_2	L2 Cache Line Size	64 bytes
l_3	L3 Cache Line Size	64 bytes
α_{mem}	Memory Latency	300-500 cycles
$\alpha_{network}$	Network Latency	
$\beta_{network}$	Inverse Network Bandwidth	

5.4.3 Test Problem Description

Four test problems were constructed in order to gather performance measurements. The test problems included a basic case, and then three other cases that refine a specific component of the mesh, e.g. spatial mesh, rays, and angles. The basic model consists of a 4x4x4 array of 1 cm^3 ray tracing modules. Each ray tracing module has 360 flat source regions. The S_8 quadrature was used with 0.05 cm ray spacing. The fine ray case used a 0.01 cm ray spacing. The fine angle case used the S_{10} quadrature, and the fine space case had 1440 flat source regions per module. Each run averaged measurements of 105 sweeps. The problem size parameters for each test case are given in Table 5.6.

Table 5.6 – Test problem size parameters

Parameter	Default	Fine Angle	Fine Ray	Fine Space
nreg	23,040	23,040	23,040	92,160
nangoct	10	15	10	10
nlongray	1,405,760	2,159,680	3,948,288	1,405,760
nseg	50,329,088	77,776,384	142,061,056	69,005,824

5.4.4 Validation of FLOP and Load Counts

The executable code was run in serial for the validation of the kernel FLOP and load counts. The code was compiled using optimizations because it was observed that the number of loads that were measured varied by several orders of magnitude; with the non-optimized code having more loads. This was investigated for several simple statements in the source code. For example, in a statement such as `a=b+c`, the measurement system with optimized code would count two loads and one store, while the non-optimized code would measure three loads and three stores. It was also observed that one of the primary optimizations performed by the compiler is minimizing load and store operations. Therefore, the optimized code was used to more consistently measure the loads based on how they were counted to produce the values in Table 5.2.

The comparison of measured FLOP and load counts averaged per sweep from the four test cases are given in Table 5.7 and Table 5.8, respectively; with the estimated theoretical values based on the expressions in Table 5.2 and the problem size values of Table 5.6. In general, the measured results in these tables must be adjusted for overhead and the adjustment calculation is shown by Eq. (5.21), where N_{raw} is the raw counts, N_{sample} is the number of measurement samples per sweep, and C_{over} is the overhead per sample for the given metric:

$$N_{adjusted} = N_{raw} - N_{sample} \times C_{over}. \qquad \textbf{Eq. (5.21)}$$

The measurement system incurs no overhead in measuring FLOPs ($C_{over,FLOP} = 0$), so the adjustment of the measured values does not change when counting FLOPs. As can be seen from the data, the expressions for the FLOP counts developed in Section 5.3 and summarized in Table 5.2 are exact for each component. In terms of the overall number of FLOPs per sweep there is a slight difference between measurement and estimated values. This is due to the fact that the component description of Eq. (5.5) does not account for all FLOPs in the actual routine. However, those FLOPs being ignored are negligible. Thus, the analytic expressions for FLOP counts as a function of problem size appear to be validated with measurement.

Table 5.7 – Comparison of measured and estimated FLOP counts for 3-D MOC kernel

Operation	Case	Measured	Estimated	Rel. Difference
build	Default	50,329,088	50,329,088	0.00%
	Fine Angle	77,776,384	77,776,384	0.00%
	Fine Rays	142,061,056	142,061,056	0.00%
	Fine Space	69,005,824	69,005,824	0.00%
exp	Default	150,987,264	150,987,264	0.00%
	Fine Angle	233,329,152	233,329,152	0.00%
	Fine Rays	426,183,168	426,183,168	0.00%
	Fine Space	207,017,472	207,017,472	0.00%
BC	Default	0	0	0.00%
	Fine Angle	0	0	0.00%
	Fine Rays	0	0	0.00%
	Fine Space	0	0	0.00%
MOC	Default	402,632,704	402,632,704	0.00%
	Fine Angle	622,211,072	622,211,072	0.00%
	Fine Rays	1,136,488,448	1,136,488,448	0.00%
	Fine Space	552,046,592	552,046,592	0.00%
scal	Default	1,843,200	1,843,200	0.00%
	Fine Angle	2,764,800	2,764,800	0.00%
	Fine Rays	1,843,200	1,843,200	0.00%
	Fine Space	7,372,800	7,372,800	0.00%
BCUp	Default	0	0	0.00%
	Fine Angle	0	0	0.00%
	Fine Rays	0	0	0.00%
	Fine Space	0	0	0.00%
flux	Default	92,160	92,160	0.00%
	Fine Angle	92,160	92,160	0.00%
	Fine Rays	92,160	92,160	0.00%
	Fine Space	368,640	368,640	0.00%
sweep	Default	605,907,578	605,907,456	2e-4%
	Fine Angle	936,196,790	936,196,608	2e-4%
	Fine Rays	1,706,691,194	1,706,691,072	7e-5%
	Fine Space	835,903,610	835,903,488	1e-4%

Table 5.8 summarizes the load counts for each component. Previously, the coefficient c_{build} for determining the number of loads in the *build* component could not be determined exactly for the general case. For that reason, these values are back-calculated from the measurements obtained for each case, and the average of the cases is 10.46 with a standard deviation between cases of 1.07 loads. These values would indicate that assuming that the number of loads is proportional to the number of segments is a reasonable assumption, provided the modularity of the problem is constant. A value of 10.5 is used as the assumed value for c_{build} for the test problems, which falls in the

expected range since it is larger than the theoretical lower bound of 3. The load counts for the *build* component and the integral counts of the kernel are given in Table 5.9.

Table 5.8 – Comparison of measured and estimated Load counts for 3-D MOC kernel

Operation	Case	Measured	Estimated	Rel. Difference
build	Default	528,904,622	$c_{build} \times \text{nseg}$	---
	Fine Angle	817,008,179	$c_{build} \times \text{nseg}$	---
	Fine Rays	1,666,736,866	$c_{build} \times \text{nseg}$	---
	Fine Space	628,680,880	$c_{build} \times \text{nseg}$	---
exp	Default	322,573,693	30,197,4528	-6.39%
	Fine Angle	489,187,877	466,658,304	-4.61%
	Fine Rays	924,320,105	852,366,336	-7.78%
	Fine Space	429,281,428	414,034,944	-3.55%
BC	Default	46,801,877	11,246,080	-75.97%
	Fine Angle	107,663,972	17,277,440	-83.95%
	Fine Rays	233,480,800	31,586,304	-86.47%
	Fine Space	36,340,420	11,246,080	-69.05%
MOC	Default	631,671,872	603,949,056	-4.39%
	Fine Angle	984,663,869	933,316,608	-5.21%
	Fine Rays	1,823,083,997	1,704,732,672	-6.49%
	Fine Space	862,377,147	828,069,888	-3.98%
scal	Default	2,124,028	1,843,200	-13.22%
	Fine Angle	3,403,315	2,764,800	-18.76%
	Fine Rays	2,014,059	1,843,200	-8.48%
	Fine Space	8,445,750	1,843,200	-78.18%
BCUp	Default	6,290,129	5,623,040	-10.62%
	Fine Angle	9,633,163	8,638,720	-10.32%
	Fine Rays	17,195,244	15,793,152	-8.15%
	Fine Space	6,390,347	5,623,040	-12.01%
flux	Default	95,547	92,160	-3.54%
	Fine Angle	97,549	92,160	-5.52%
	Fine Rays	97,309	92,160	-5.29%
	Fine Space	384,086	368,640	-4.02%

There appears to be a general trend of under-predicting the number of loads for the load counts of the other components listed in Table 5.8. Because this is a systematic trend, the cause of this may be due to the way the overhead of the measurement is estimated. The predicted loads agree within 10% of the measurement for the *exp*, *MOC*, *ray*, and *flux* components. The difference of measurement and estimated load counts for *BCUp* and *scal* is notably higher, with the fine space case of the *scal* component having an extremely large difference. It is unclear why this one case has such a large difference,

but it may be that the load of the machine was abnormal for that measurement. The *BC* component is consistently and significantly under-predicted. It is very likely that the method for estimating the overhead is not valid for this component. Since the component represents such a small set of operations (4 loads per measurement), the relative overhead is much larger than the number of operations that are expected to be measured. Therefore, it was concluded that the resolution of the measurement system is not fine enough to accurately obtain measurements for *BC* component.

Table 5.9 shows the measured load counts for the build component and the whole kernel, along with the sum of each component. The sum of the measurements for each component agrees well with the sum of the estimated loads for each component. However, there is a definite bias in the sum of the measurements of each component compared to the integral measurement of the 3-D MOC kernel. Similar to the case with the FLOP counts, this is likely due to not capturing all the load operations of the kernel in the component model of Eq. (5.4). In the case of the load counts, the differences are notably higher than they are for the FLOP counts, though the data suggests that approximately 5%-10% of the loads that occur within the kernel are not captured with the component model. Some of this difference may also be due to the manner in which the compiler optimizes the instrumented code.

Table 5.9 – Comparison of measured and estimated Load counts with c_{build} = 10.5

Operation	Case	Measured	Estimated	Rel. Difference
build	Default	528,904,622	528,455,424	-0.08%
	Fine Angle	817,008,179	816,652,032	-0.04%
	Fine Rays	1,666,736,866	1,491,641,088	-10.51%
	Fine Space	628,680,880	724,561,152	+15.25%
sweep	Default	1,703,277,947	1,453,206,528	-14.68%
	Fine Angle	2,629,365,246	2,245,423,104	-14.60%
	Fine Rays	4,921,813,948	4,098,100,992	-16.74%
	Fine Space	2,204,301,869	1,985,839,104	-9.91%
Sum of components	Default	1,538,461,768	1,453,206,528	-5.54%
	Fine Angle	2,411,657,923	2,245,423,104	-6.89%
	Fine Rays	4,548,577,055	4,098,100,992	-12.19%
	Fine Space	1,954,089,078	1,985,839,104	+0.70%

In summary, the measurement system confirms the methodology of estimating the FLOP count. The estimated load counts have some non-trivial differences, and it is

considerably more difficult in this case to obtain accurate measurements of the number of loads. Most of these differences can be attributed to not being able to accurately estimate the measurement overhead for all components. However, in the subsequently analyses the component model is only used to obtain the integral number of loads for the entire kernel, and the differences in the estimated and measured integral load counts is roughly 10%. Thus, while the differences between the model and measurement may be quite large for some components (80%), the differences for the kernel are considerably smaller. This difference in the integral loads will be treated as an uncertainty in the number of loads, since the root cause is an uncertainty in the measurement overhead. Fortunately, this uncertainty is bounded by the upper and lower cache miss values, and so this 5%-10% difference in the integral counts should be acceptable for providing an upper and lower performance bound of the 3-D MOC kernel.

5.4.5 Validation of Execution Time in Serial

Having established some level of consistency between the measurements and expected counts for at least loads and FLOPs, the focus can now move to the prediction of execution time and performance. First, the measured execution time is compared to the execution time computed from Eq. (5.3) using the measured values of FLOPs, loads, and cache misses, and the hardware values of Table 5.5. Because the L3 cache misses could not be measured directly, their values were estimated as 10% of the number of L2 misses. Assuming no L3 misses changes the relative difference between the measured and computed execution time by roughly 0.5%, so the effect of L3 misses on the overall differences is minimal.

The measured average sweep time and computed sweep time for non-optimized and optimized executables are compared in Table 5.10 and Table 5.11, respectively. The differences in the measured and calculated execution time is just over 10% for the non-optimized case. This can be attributed to one of two things: the accuracy of the hardware values of Table 5.5, or the fact that an assumption was made as to the number of L3 misses, since this value could not be measured. Since the assumption about the L3 misses has minimal effect, the majority of the difference can be attributed to uncertainties in L1

cache access time and to the time to execute a FLOP. The L2 access time also plays a role, but it is less important.

In the optimized case, the agreement is slightly better and is within 10%. The predicted execution time for the fine ray case is actually higher than the measured time, suggesting that some additional performance gain is achieved through optimizations by decreasing the apparent time for some hardware event (either a FLOP or cache access), which is not unusual. Since the use case for performance will be the optimized executable, this analysis suggests that the model should be able to predict the execution time to within 8%, of the actual time. This is reasonably accurate for the prediction of the execution time, and suggests that the methodologies used to obtain the model hardware values for the evaluation of Eq. (5.3) are acceptable, although could be slightly improved.

Table 5.10 – Comparison of measured and computed execution times with no optimizations

Case	Measured Execution Time	Computed Execution Time	Rel. Difference
Default	7.1835	6.2129	-13.51%
Fine Angle	11.0630	9.5784	-13.42%
Fine Ray	20.5691	18.1020	-11.99%
Fine Space	9.7376	8.4605	-13.11%

Table 5.11 – Comparison of measured and computed execution times with optimizations

Case	Measured Execution Time	Computed Execution Time	Rel. Difference
Default	1.2991	1.2029	-7.41%
Fine Angle	2.0150	1.8663	-7.38%
Fine Ray	3.8637	4.0357	+4.45%
Fine Space	1.7228	1.6902	-1.89%

The relative contribution to total measured and calculated execution time of the various components for the non-optimized executable are shown in Figure 5.6 and Figure 5.7, respectively. Somewhat surprisingly, the component that builds the long ray data is one of the most expensive parts of the kernel, which suggests that this area should be addressed in order to improve the performance of this kernel in the future. This is also likely true of most other MOC kernels, whether they are 2-D or 1-D.

As for the agreement between the measured relative execution times and the predicted execution times, the largest difference in the predicted percentage occurs for the *build* component, with a magnitude of 7%. This is primarily due to inaccuracy in

some of the ratios of the average values for the model hardware data. These ratios for the model values (e.g. timer per FLOP to time per L1 access) would need to be adjusted in order to achieve better agreement between the model and measured relative execution times. The components based on the number of regions have the smallest contribution, which is consistent with the observation that this value is several orders of magnitude less than the number of segments or number of long rays. The other observation is that the ratios between the different components do not change significantly between the different cases; this suggests that the ratio of segments to long rays does not vary much between these cases.

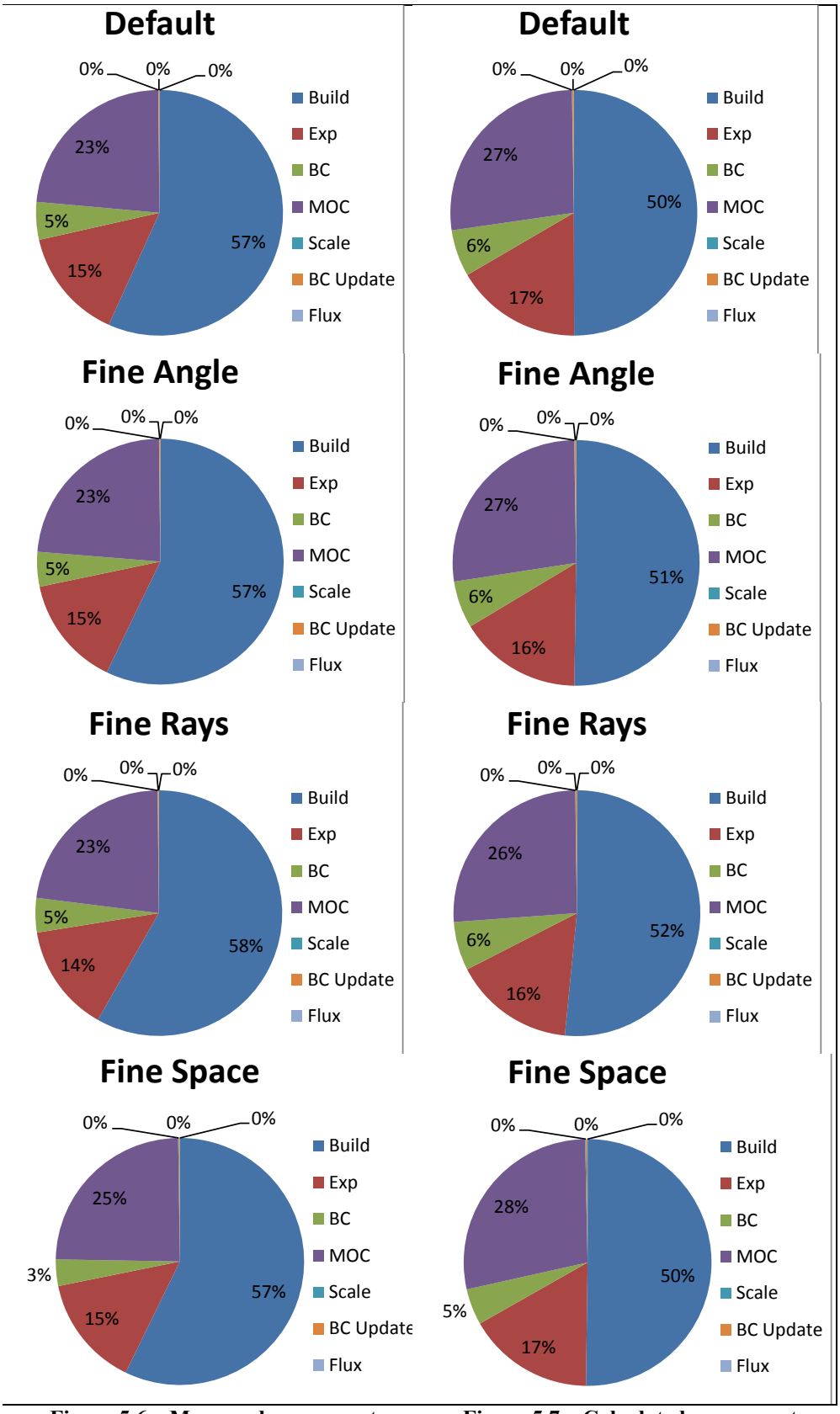


Figure 5.6 – Measured component relative execution times

Figure 5.7 – Calculated component relative execution times

5.4.6 Baseline Performance in Serial

Table 5.12 shows the measured performance for the kernel and estimated upper and lower bounds. The measured performance is within the estimated bounds, therefore the performance model and estimated upper and lower bounds are assumed to be valid for predicting the performance of the kernel as a function of the problem size and machine hardware characteristics.

Table 5.12 – 3-D MOC kernel performance in serial on Sunspear

Case	Measured (MFLOPS)	Lower Bound (MFLOPS)	Upper Bound (MFLOPS)	Realized Fraction of Upper Bound
Default	466.4	0.467	897.0	52.0%
Fine Angle	464.6	0.469	897.0	51.8%
Fine Rays	441.7	0.468	896.2	49.3%
Fine Space	485.2	0.166	904.1	53.7%

Because only about 50% of the theoretical upper bound is realized, this suggests that some improvements may be possible to further reduce cache misses. The theoretical peak performance of one core on Sunspear is 10.4 GFLOPS, which means that the initial performance of the kernel is getting roughly 4.5% of this peak.

The upper bound on the kernel performance neglects all cache misses and only counts L1 accesses. The upper bounds in Table 5.12 used execution times based on 1 FLOP/cycle and 1 load/cycle; the ~900 MFLOPS predicted as the upper bound on performance is approximately 9% of the processor's theoretical peak. If Eq. (5.3) is simplified to only include the time for stores, then it can reduced to:

$$T = Ft_f\left(1+\frac{\alpha_1}{t_f}\frac{L}{F}\right). \qquad \textbf{Eq. (5.22)}$$

Eq. (5.22) provides some insight into the maximum possible performance expected from the MOC kernel. For the 9% of peak performance estimated for the MOC kernel the α_1/t_f factor is 1.0. To achieve at least half of the peak performance, an algorithm would need an L/F ratio of 1.0. From data in Table 5.2, it is estimated that the L/F ratio for the kernel is at least 2.0. With this value, the maximum fraction of the peak performance achievable would only be 33%. In actuality the L/F of the problems tested is

probably closer to 9.0, implying that the observed computational intensity of the kernel is quite low at ~0.10. This also suggests some possible improvement to the kernel through reducing the L/F ratio. Through further analysis of Eq. (5.22), it can be deduced that an architecture that gives better performance would mean that the machine balance, α_1/t_f, would need to be less than 1.0. Unfortunately, for the assumed target architecture, this is never the case, and does not appear to be the case for any near term architectures.

5.5 Sensitivity of Serial Performance to Hardware Characteristics

In this section the performance model of Eq. (5.4) and Eq. (5.5) is examined for a range of hardware values for the memory access latencies and FLOP execution times. For this analysis, the assumed problem size is the default test case described in Section 5.4.3. Other simplifying assumptions are used based on the measurement results of Section 5.4. Instead of using the theoretical upper and lower bounds of the cache misses, the cache misses at each level are based on those observed for the default test case. In this case, M_1 was approximately 8.43% of the number of loads, and M_2 was 0.07% the number of loads. For the analysis M_1 is assumed to be 8.5% of the number of loads and for other levels of cache $M_j = 0.01M_{j-1}$ for $j > 1$ is assumed; α_{mem} is assumed to be 1000 ns.

The first sensitivity examined is the model's sensitivity to t_f and α_1 and levels of cache, κ. The predicted performance for each case is shown as a contour plot in Figure 5.8 through Figure 5.11 for the different values of κ. The cache latencies for levels of cache are assumed where the access latency for each level $\alpha_j = 10\alpha_{j-1}$ for $j > 1$.

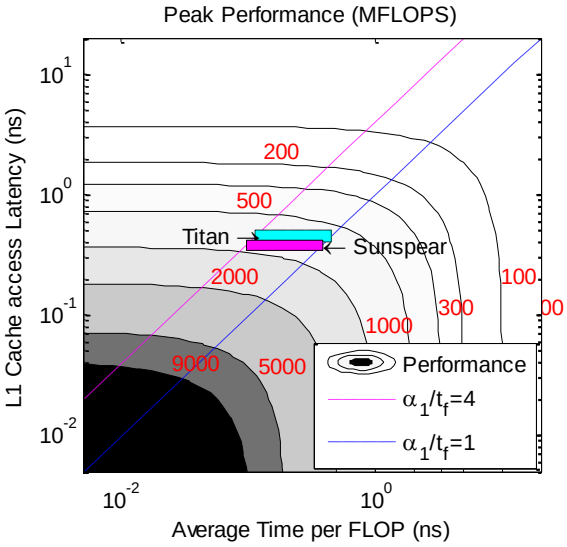


Figure 5.8 – Sensitivity of peak performance to α_1 and t_f

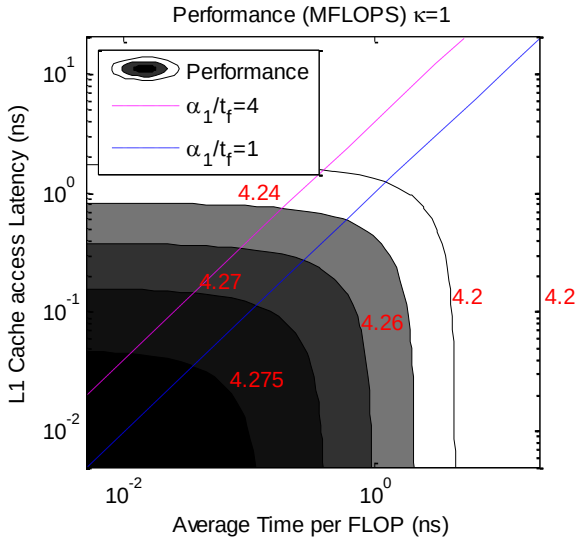


Figure 5.9 – Sensitivity of performance to α_1 and t_f for single-level cache

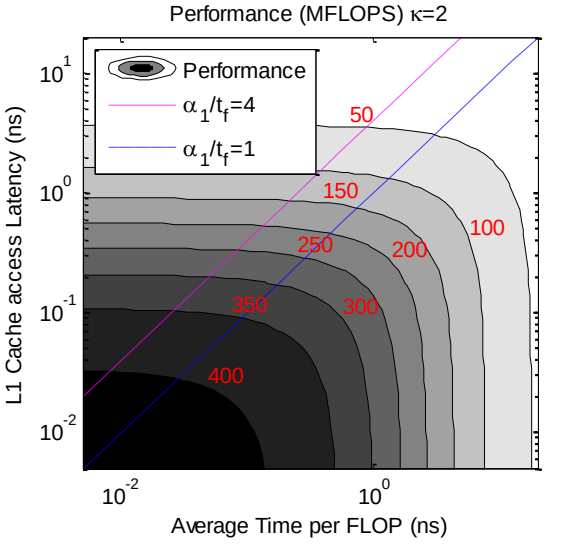


Figure 5.10 – Sensitivity of performance to α_1 and t_f for two-level cache

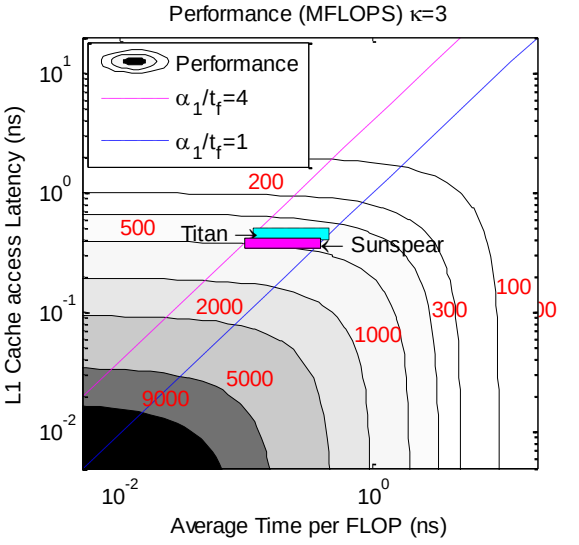


Figure 5.11 – Sensitivity of performance to α_1 and t_f for three-level cache

The data in Figure 5.8 is consistent with the theoretical upper bound (zero cache misses) of performance for the 3-D MOC kernel. Several lines are overlaid on the contours to show when the performance becomes more sensitive to a particular hardware property. The region above the magenta line describes an architecture in which the time per flop is at least 4x faster than the time for a memory access. In this region it is observed that the performance is more sensitive to the average memory access time, and the performance of the kernel will be limited by this hardware characteristic. The other

line (blue) in Figure 5.8 describes when the time per flop and L1 cache access time are equivalent. The region below this line is indicative of an architecture in which the cache access time is faster than the time per flop, which is generally not the type of architecture that is manufactured. However, for an architecture of this type the performance would be dictated by the time per flop. The region between the two lines is fairly indicative of most commodity architectures, and in this region the performance is generally more sensitive to cache access time compared to the time per flop. The magenta and cyan regions denote where the hardware characteristics of test machines used in this chapter and Chapter 6 were measured.

In Figure 5.9 the performance does not really change with either hardware property, and instead is limited by the 1 μs access time to retrieve data form main memory. Figure 5.10 shows the performance sensitivity for an architecture with a two level cache. Here the performance is still severely limited by the access time to main memory compared to the peak performance of Figure 5.8. However, there is some sensitivity to the time per flop and the cache access latency. The sensitivities for the two level cache basically behave in a similar manner as peak performance case, although the magnitude of these sensitivities is reduced considerably.

With the three level cache architecture shown in Figure 5.11, the performance sensitivity to the hardware properties is very similar to that observed in Figure 5.8. However, for a given (t_f,α_1) pair, the predicted performance in Figure 5.11 is approximately half of Figure 5.8, which is consistent with Table 5.12. This would indicate that, assuming the cache misses are reduced by a factor of 100 at each level and the memory access times are increased by only a factor of 10, then this becomes a very good architecture for achieving behavior similar to the peak performance of the algorithm. Although it is not shown, the case of an architecture with a four level cache was also investigated and it was observed to have almost exactly the same performance as the architecture with three cache levels.

The performance sensitivities are also examined for the L1 and L2 cache access times. In this study the time per flop and main memory access times are assumed to be constant at 0.2 ns and 1 μs, respectively. A contour plot of the performance as a function of these two hardware properties is shown in Figure 5.12. Again, the test machine

hardware characteristics are highlighted by magenta and cyan boxes that are labeled with the machine names. In this figure, the region below the magenta dashed line can be ignored. This region would indicate where the L2 cache access time is faster than the L1 cache access time. The blue line divides the plot into two regions. The region above the line denotes where the performance is limited by the L2 cache access time, and conversely below this line is where the performance is limited by the L1 cache access time. The sensitivity of the performance with respect to the L1 cache access time is 5x higher than the L2 access time.

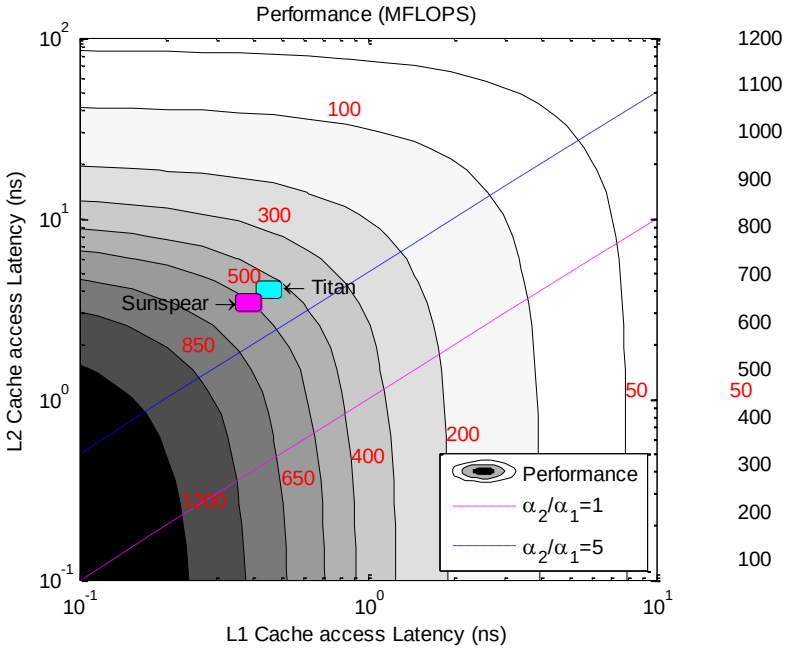


Figure 5.12 – Performance as a function of L1 and L2 cache latencies

5.6 Summary

This chapter presents a conceptual model of the target architecture and the basic equations for a latency based performance model. Expressions for the FLOP and load counts for the 3-D MOC kernel were developed in terms of key problem size parameters. The performance model was validated for a specific machine whose architecture fits within the conceptual model, and using four small problems with representative meshing. This required methods for determining the performance model hardware coefficients for the test machine. The measurement system was used to validate the expressions for FLOP

and load counts, and then to validate the execution time model. The FLOP counts are exact, while the load counts and execution time may have uncertainties of up to 10%. The baseline performance of the serial kernel was then given, and the measured performance was shown in comparison to the model performance bounds. This provides a solid foundation for the next chapter in which the performance model analysis of the parallel 3-D MOC kernel is described. Finally, a parametric study of the performance as a function of the hardware properties is presented to indicate the types of architectures that are likely to get the best performance, and which properties have the greatest influence on performance.

Chapter 6

PERFORMANCE ANALYSIS OF THE PARALLEL 3-D MOC KERNEL

In this chapter the performance model developed in Chapter 5 is applied to the parallel execution of the MOC kernel. First, the parallel efficiency and speedup are defined with respect to two types of scaling metrics. The methods are then described for determining the model hardware coefficients for the network latency and bandwidth required by the parallel execution time models. In order to show that the performance model developed in Chapter 5 is still valid when extended to the parallel domain, a section is included on the experimental validation of the performance model. The performance model is then analyzed, given a range of input parameters, to show the sensitivity of the algorithm to the hardware coefficients. A parametric study is also presented to show the efficacy of the various decomposition strategies and the preferred optimum decomposition for a given problem. Finally, the results and conclusions of these analyses are summarized.

6.1 Parallel Performance Metrics

In the previous chapter the performance of the 3D MOC kernel for serial execution was quantified with respect to the processor's theoretical peak performance. In parallel computing, other metrics are typically used to quantify and describe the performance of an algorithm. Of primary importance is how well the algorithm scales, or how well it performs as more processors are added to a calculation. For a parallel algorithm there are two notions of scaling: strong scaling and weak scaling. *Strong scaling* refers to how the solution time varies when the input problem size is fixed and the number of processors is varied. For strong scaling, two metrics are often used to quantify the performance;

namely the speedup and efficiency. The speedup, S, is defined by Eq. (6.1) and the efficiency, E_{strong}, is defined by Eq. (6.2); where T is the time, P_{size} is the problem size, and N_p is the number of processors. Strong scaling is indicative of how finely grained an algorithm may be executed in parallel, and how much overhead exists for the parallelism relative to the actual computation.

$$S\left(P_{size}, N_p\right) \equiv \frac{T\left(P_{size},1\right)}{T\left(P_{size}, N_p\right)}, \qquad \textbf{Eq. (6.1)}$$

$$E_{strong}\left(P_{size}, N_p\right) \equiv \frac{T\left(P_{size},1\right)}{N_p \times T\left(P_{size}, N_p\right)} = \frac{S\left(P_{size}, N_p\right)}{N_p}. \qquad \textbf{Eq. (6.2)}$$

For the *weak scaling* of an algorithm the work per process is fixed and the number of processors is varied. The weak scaling indicates whether the parallel overhead varies faster or slower than the amount of work as the problem size increases, and is relevant in determining the overall size of a problem that can be solved efficiently. The primary metric for weak scaling is the *parallel efficiency*, given below in Eq. (6.3), which has a slightly different definition than Eq. (6.2).

$$E_{weak}\left(P_{size}, N_p\right) \equiv \frac{T\left(P_{size},1\right)}{T\left(P_{size} \times N_p, N_p\right)}. \qquad \textbf{Eq. (6.3)}$$

In the following analyses of Section 6.3 and 6.4 the expression for T in the above equations is the T_{sweep} of Eq. (5.18) derived in Chapter 5.

6.2 Experimental Evaluation of Parallel Performance Model

For the experimental validation of the performance model in parallel, the specific components related to the parallelism, T_{ray}, T_{ang}, T_{space}, are individually measured and compared to the model. In validating these components, the test problem used in the default case described in Chapter 5 is again used. The measurement system used to collect data is the same as that described in Section 5.4.1. However, the machine used to run the performance tests is different than the one used in Chapter 5. For these

performance tests Titan [54] is used, which is a Cray XK7 having a single AMD Opteron™ 6274 processor and 32GB of RAM on each compute node. The AMD Opteron™ 6274 is a 16-core processor of the same class as the one in Sunspear, but the clock speed is 2.2GHz. The node architecture on Titan is similar to the illustration in Figure 5.3, but there are 8 core-pairs instead of 6. Titan has 18,688 physical compute nodes, which are connected using Cray's Gemini™ interconnect.

6.2.1 Determining Network Model Hardware Coefficients

The cache latencies and time per flop are determined using the same methodology as in Chapter 5. The hardware coefficients for the network needed to evaluate Eq. (5.8) and Eq. (5.13) or Eq. (5.14), are determined using another suite of micro benchmarks. The OSU micro-benchmarks for MPI (OMB MPI tests) [55], are used for this purpose. The OMB MPI tests evaluate multiple communication patterns on a network to measure the network latencies and bandwidth. Three of the benchmark suite's tests were executed: the latency test, the bandwidth test, and the all reduce test.

The latency test characterizes the time required for a message to travel from point-to-point. The benchmark measures the time required for a message to be sent from one processor to another and then back, and then reports the average time for a one way communication. This is done for various message sizes, and the resulting output should fit Eq. (5.8), provided that $\alpha_{network}$ and $\beta_{network}$ are known. The test is also performed with a message size of 0, and this is the case used to determine $\alpha_{network}$. This benchmark was used to measure the point-to-point communication time between the physical nodes, the two NUMA nodes on a single physical node, and two core pairs within a NUMA node. This is because it will not necessarily be known which processors will be using which communication layer at run time, since it could potentially be any of the three. The maximum message size measured was 32 MB. The benchmark output for Titan is shown in Figure 6.1 and includes the average of three runs.

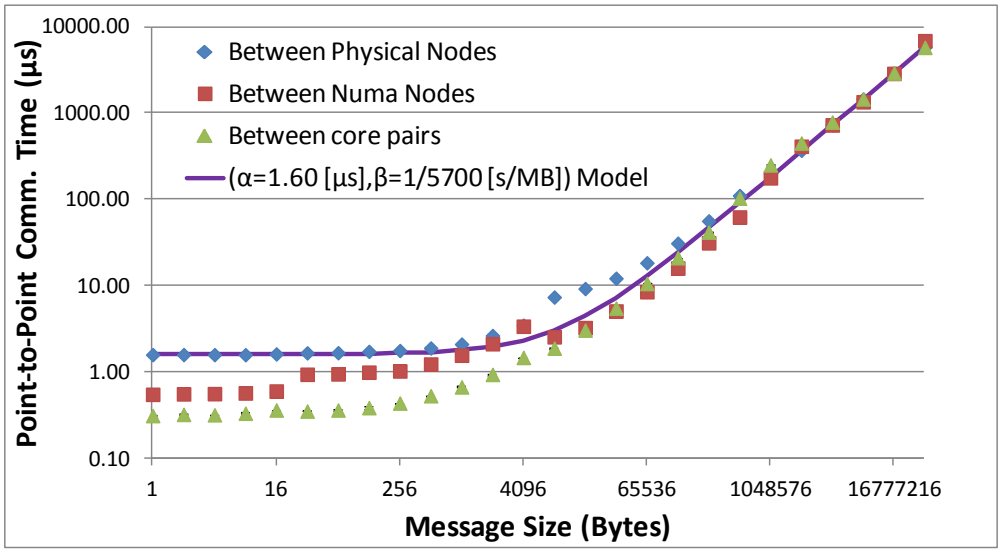


Figure 6.1 – Output of OMB point-to-point latency test

There are error bars for the standard deviation, but they are so small that they are not visible in Figure 6.1. The maximum relative standard deviation observed was 3.98%, with the between node communication having the highest standard deviation. The measured latency was 1.57 µs between physical nodes, 0.53 µs between NUMA nodes, and 0.28 µs between cores in a core pair.

The bandwidth test determines the network's bandwidth, which measures the data throughput for a given time window using multiple message sizes. The number of concurrent messages is fixed, so as the message size is increased, the measured bandwidth should become asymptotic as the network hardware becomes saturated. Again, this benchmark was used to measure the bandwidth of the three communication layers, and the results reported are the average of three trials. A maximum message size of 32 MB was also used in this test. The results of this benchmark are shown in Figure 6.2. In this figure the measured bandwidth appears to go asymptotic in the range of 5000 MB/s and 6400 MB/s. However, the NUMA node bandwidth exceeds this by nearly a factor of two for a limited range of message sizes before decreasing to the indicated range.

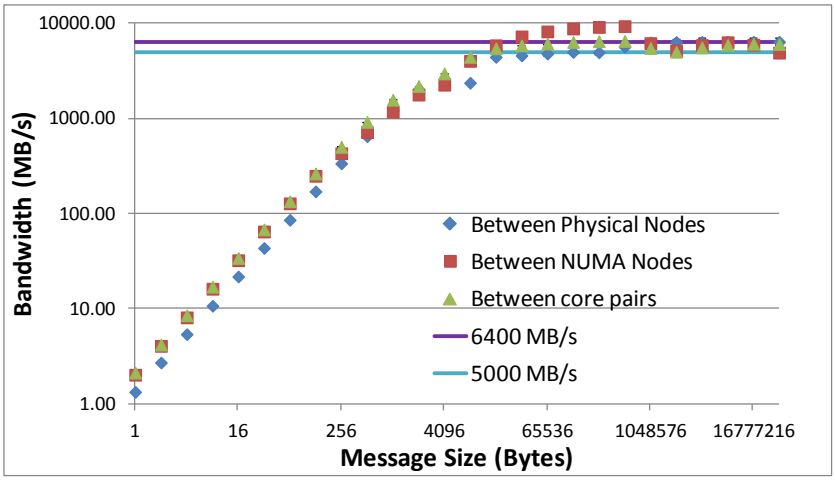


Figure 6.2 – Output of OMB point-to-point bandwidth test

The all reduce micro benchmark measures the time to perform an all reduce operation for various message sizes and reports the average of all processors involved in the operation. This benchmark was executed for message sizes up to 32 MB and used processor counts of 1 through 8, and then powers of two afterward up to 512 processors. The results of the benchmark are shown in Figure 6.3.

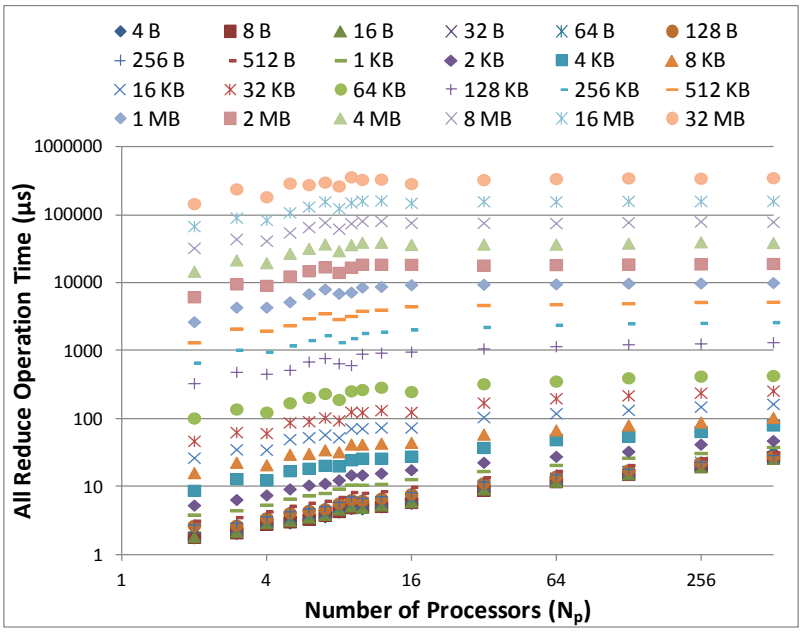


Figure 6.3 – Output of OMB all reduce test

As noted in [50] an MPI library may include multiple algorithms, and in order to achieve optimal performance, different algorithms will be chosen based on the message size and number of processors. Furthermore, multiple algorithms may be used for different parts of the communication in the all reduce operation. Thus, unless the exact implementation of the all reduce operation is known, it is difficult to rely on a algorithm specific model, such as those given in Eq. (5.13) or Eq. (5.14). Nonetheless, these models may be generalized into a semi-empirical formula, and the remaining coefficients may be determined through analysis of the data in Figure 6.3. The execution time for the all reduce operation may be generalized from of Eq. (5.13) and Eq. (5.14) and expressed in the form shown in Eq. (6.4). Here c_1 and c_2 are the general coefficients, which are functions of the number of processors, that may be fit to an implementation of all reduce when the algorithm used is not known. The rest of the terms in Eq. (6.4) have the same meaning as in Eq. (5.13) and Eq. (5.14), although γ' has a slightly different meaning. This is illustrated by comparing Eq. (5.14) to Eq. (6.4) and noting that $\beta_{network}$ and γ have different coefficients in Eq. (5.14). However, since it is difficult to measure γ directly, this term is allowed to be adjusted by a constant factor to fit the semi-empirical model of Eq. (6.4), hence the use of the γ' notation. In Eq. (6.4), the functions for the coefficients c_1 and c_2 are likely to be non-linear.

$$T_{allreduce}\left(N_p, N\right) = c_1\left(N_p\right)\alpha_{network} + c_2\left(N_p\right)\left(\beta_{network} + \gamma'\right)N. \qquad \textbf{Eq. (6.4)}$$

First, the coefficient of the latency, c_1, is determined. This can be done using the benchmark output for the cases with the smallest message size and varying processor counts, since the execution time of these cases will be latency dominated. Figure 6.4 shows this data along with two different expressions for c_1. The measured data clearly has a non-linear trend when plotted on the $\log_2(N_p)$ scale, which is contrary to the latency coefficients proposed by Eq. (5.13) and Eq. (5.14). It is however remarkably well fit by the expression $\sqrt{N_p}$.

It is less straightforward to determine the remaining coefficients c_2 and γ', since it is not possible to separate the $\left(\beta_{network} + \gamma'\right)$ term from c_2 using the data collected from

the benchmark. Therefore, the approach used is to assume that the $(\beta_{network} + \gamma')$ term is an unknown constant, and then to determine an expression for c_2 to the measured data by scaling this expression until the measured data is reasonably well fit. This still requires the correct functional dependence for c_2, even though it is allowed to be scaled. Once c_2 is known, γ' can be determined from the unknown constant and $\beta_{network}$ can be determined as previously described.

In order to determine c_2, it is necessary to use a case with a sufficiently large message so the influence of the latency on the operation time can be minimized. Then the operation time may be divided by the message size, which leaves the term of interest. These values are then plotted as a function of the number of processors. The 32 MB message size case is shown in Figure 6.5, along with some possible expressions for c_2. The $(N_p\text{-}1)/N_p$ curve fits the trend in the measurement data fairly well. Selecting $(\beta_{network} + \gamma')$ to be 0.011 μs/B bounds the data and minimizes the error for large processor counts, but the value of 0.0095 μs/B fits the data better for lower processor counts. The difference here is likely related to the different hardware involved with communication on the node and between the nodes. This supported by the observation that the measured data seems to have a jump right at 16, which is the number of cores on a node.

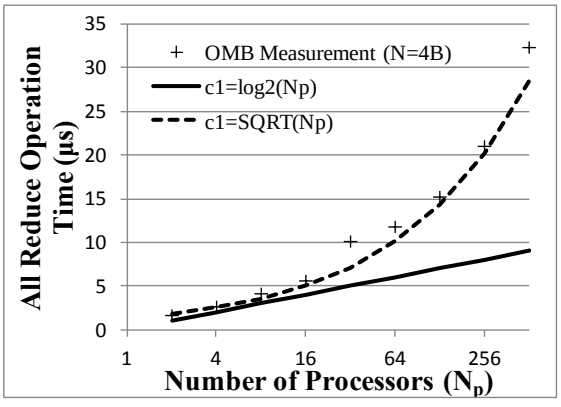


Figure 6.4 – Curve fit for $c_1(N_p)$

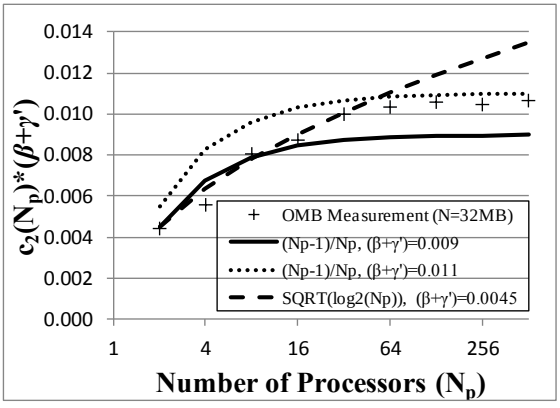


Figure 6.5 – Curve fit for $c_2(N_p)$

From the analysis of these benchmarks, the final hardware values selected for Titan in order to evaluate the performance models are given in Table 6.1, and the model

used to evaluate the all reduce operation is given in Eq. (6.5), which applies when N_p is a power of 2.

Table 6.1 – Performance model hardware values for Titan

Symbol	Name	Value
C	Clock Speed	2.2 GHz
t_f	time per FLOP	0.25-1 FLOPs/cycles
α_1	L1 Cache Latency	~1 cycle
α_2	L2 Cache Latency	~8-10 cycles
α_3	L3 Cache Latency	~20-110 cycles
l_1	L1 Cache Line Size	64 bytes
l_2	L2 Cache Line Size	64 bytes
l_3	L3 Cache Line Size	64 bytes
α_{mem}	Memory Latency	300-500 cycles
$\alpha_{network}$	Network Latency	~0.25-1.6 μs
$\beta_{network}$	Inverse Network Bandwidth	~1.49e-4 - 1.91 e-4 μs/B
γ'	Computation cost ber byte	~0.009309 - 0.010851 μs/B

$$T_{allreduce}(N_p, N) = \sqrt{N_p}\,\alpha_{network} + \frac{N_p - 1}{N_p}(\beta_{network} + \gamma')N. \qquad \textbf{Eq. (6.5)}$$

6.2.2 Determining OpenMP Run-time Library Overhead

The final set of unknown expressions that must be determined to evaluate the parallel performance model involve the ray decomposition from Eq. (5.20). These terms account for the overhead of the OpenMP run-time library routines, and in general these are not known exactly. The general approach is to determine the parameters for a specific library and architecture, and assume they are reasonably valid for a wider range of compilers and architectures. To determine the overhead of the OpenMP routines used in the MOC kernel, another set of micro-benchmarks are analyzed on the test machine. The EPCC OpenMP Microbenchmark v2.0 suite [56] developed at the University of Edinburgh is used to evaluate the overhead of the OpenMP library for the `barrier`, `single`, and `parallel` constructs, and the various scheduling strategies of the `do` construct. The basic methodology of these benchmarks is described in detail in [57] and essentially involves the execution of a block of code with and without the OpenMP constructs. The results obtained on Titan are shown in Figure 6.6 and Figure 6.7 below.

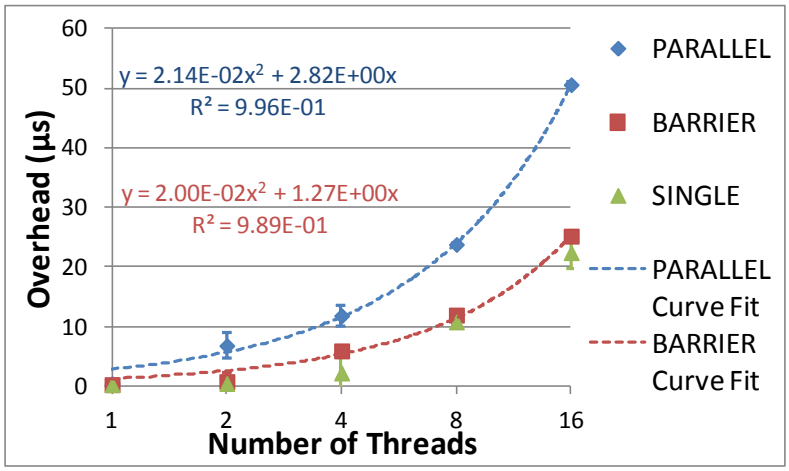


Figure 6.6 – OpenMP run-time library overhead for various constructs involving synchronization

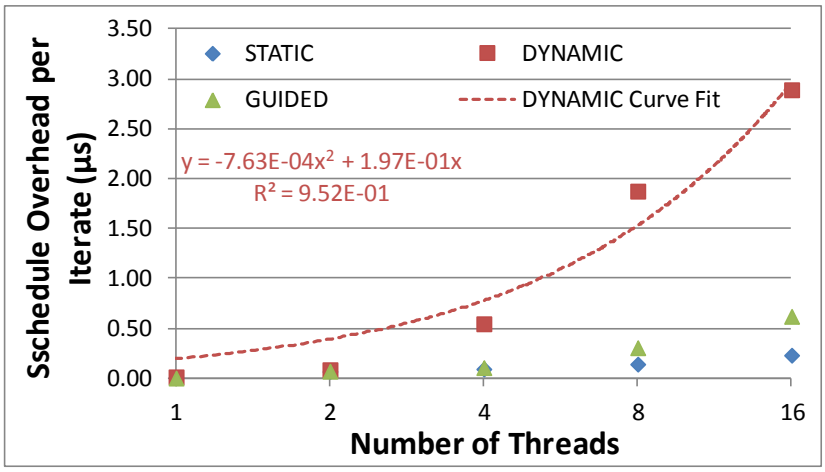


Figure 6.7 – OpenMP run-time library overhead for various scheduling algorithms

It is apparent in Figure 6.6 that all the constructs' overheads grow at a rate larger than $O(nt)$, where nt is the number of threads, so there will be definite issues in scaling the ray decomposition to large thread counts. The overhead for the `single` construct is comparable to the `barrier`; this is because the `single` construct has an implied barrier on exit. In Figure 6.7, the dynamic scheduling algorithm used by the kernel has the highest overhead per loop iteration of the scheduling methods provided by the OpenMP run time library. The overhead for the dynamic scheduling also grows at a rate larger than $O(nt)$. Linear regressions were performed to fit the data with 2^{nd} order polynomials, which could then be used in the performance model. These equations are also shown in the figures.

6.2.3 Validation of Parallel Performance Model

In this section the parallel efficiency predicted by the performance model is compared to the measured parallel efficiency for the default case test problem. The performance model for the strong scaling efficiency is evaluated against each of the ray decomposition, angle decomposition, and spatial decomposition separately. For the angle and ray decomposition 2, 4, 8, and 16 processors are used. The Titan nodes only have 16 cores, which is the reason for limiting this study to this processor count. The angular decomposition does not have good partitioning beyond 20. Furthermore the semi-empirical model for which coefficients were determined in the previous section was only evaluated for powers of two. In general, execution time models for the all reduce operation vary noticeably when non-power of two numbers of processors are used. Therefore, using this set of numbers of processors ensures a more straightforward comparison to the performance model. The spatial decomposition uses 8 and 64 processors for the strong scaling because this creates equal-sized subdomains that maintain the same surface area to volume ratio. The spatial decomposition also includes an additional component for the weak scaling.

Table 6.2 below shows the measured parallel efficiency for the strong scaling of the various decompositions and processor counts compared to the parallel efficiency predicted by the performance model.

Before assessing the comparison of measurement to the performance model, it is essential to point out that, where possible, the decompositions were mapped to the node architecture in two distinct ways. This was done to highlight the effect of a unique architectural feature of the AMD Opteron™ 6200 series processors noted previously in Section 5.4.2; namely that the chip has two integer cores that share a floating point arithmetic unit (FPU). The reasoning behind this processor design feature is that one core will be performing data fetches while the other will be using the FPU, this is a suitable assumption for data servers. However, for numerically intensive scientific software, this does affect performance. This effect becomes apparent by comparing the columns in the table labeled "by FPU" and "by core". The "by FPU" column are results measured when

only one process per core pair, or one core process per FPU is used. The columns labeled "by-core" refer to measurements taken when both cores sharing a FPU are used.

Table 6.2 – Comparison of measured and predicted parallel efficiency

N_p	Ray Decomposition			Angle Decomposition			Space Decomposition		
	by FPU	by core	Model	by FPU	by core	Model	by FPU	by core	Model
2	104.9%	53.5%	99.40%	101.8%	80.00%	99.03%	N/A	N/A	N/A
4	100.5%	67.04%	97.89%	98.18%	78.88%	97.15%	N/A	N/A	N/A
8	87.49%	67.33%	94.22%	96.05%	71.03%	93.60%	98.91%	76.32%	99.52%
16	N/A	62.17%	86.78%	89.30%	63.85%	87.22%	N/A	N/A	N/A
64	N/A	N/A	N/A	N/A	N/A	N/A	93.14%	71.66%	98.69%

Comparing these two columns of Table 6.2 it is apparent that overloading the FPU reduces the parallel efficiency for *any* parallel decomposition of the kernel by 20% to 50%. This raises questions about the most meaningful choice for the reference calculation to measure the efficiency of the parallel decomposition for this kind of processor.

To illustrate this point, one may note that the efficiency for the ray decomposition when using both integer cores sharing the FPU are ~50%-60%. However, if the reference case is taken to be the execution time of two threads sharing the FPU then the efficiencies of the cases of 4, 8, and 16 threads all increase by roughly a factor of 2 or more, and the parallel efficiency can be considered quite good. The important conclusion for the purposes here is that the performance model does not consider the effect of this architectural feature, so comparing the performance model to the "by FPU" column is more consistent.

From the comparison to the "by FPU" results it is clear that the model for the angular decomposition matches the measurement very well, having no differences in predicted efficiency larger than 3%. This indicates that the model correctly characterizes the overhead contributing to the execution time as the time to execute the call to `MPI_Allreduce`, and that the other components of the kernel are decreased linearly by the reciprocal of the number of angular domains.

For the ray decomposition, the model under-predicts the observed parallel efficiency slightly for low processor counts when the measurement shows the efficiency to be super-linear. This super-linear scaling is likely due to better cache utilization from introducing more cache with the other processors. The other observed differences between the model and the measurement can be attributed to subtleties in the hardware not captured by the latency-based performance model. For the 8 threads case in the "by FPU" column, 4 threads exist on each NUMA node, and the memory access times between NUMA nodes to the same shared memory is probably longer than memory access times to the memory shared by threads on the same NUMA node. Consequently, the performance model's ability to predict the parallel efficiency for the ray decomposition is acceptable. It lacks some important terms to capture the effects of subtleties of the hardware, but so long as careful attention is given to these details, the model agrees within 5% of the measured result.

The spatial decomposition also agrees reasonably well with the performance model for the strong scaling efficiency, but there are only two points for comparison in this test problem. The model over-predicts the efficiency compared to the measured result, which suggests that the effect the model is not accounting for that causes additional degradation in the parallel efficiency is related to having the MPI library deal with multiple outstanding messages simultaneously. The weak scaling results, shown in Figure 6.8 differ slightly from the model's prediction by a small constant factor for the "by FPU" measurement, indicating a systemic issue, probably with a hardware coefficient. Otherwise, the model is accurate in predicting the parallel performance to within about 7% of measurement. Figure 6.9 shows the "by core" results using two references, one a single core, and the other a single NUMA node. These results show the same trend for the weak scaling as the "by FPU" measurement. This again illustrates the issue of overloading the FPU and choosing a meaningful reference calculation from which to measure the efficiency.

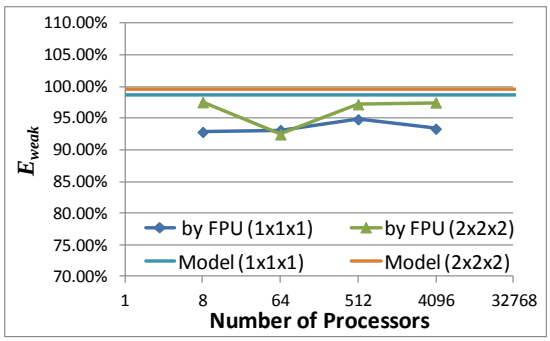


Figure 6.8 – Comparison of predicted and measured weak scaling efficiency for spatial decomposition

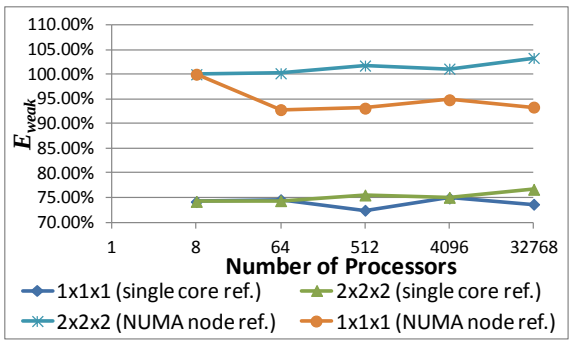


Figure 6.9 – Weak scaling efficiency for spatial decomposition with different reference cases

In conclusion, the performance model is shown to agree within 7% of the measured results for predicting the parallel efficiency for each implementation of the parallel decomposition. This is in spite of the observation that the model is shown to be deficient for capturing a subtle, yet key, architectural feature of the processors on the test machine: sharing an FPU between two cores. However, provided the measurement is made consistently with the assumption of the model, in which each processor has its own FPU, this is a non-issue. The model can also easily account for the shared FPU by using a different reference measurement, namely the time for the core pair to execute the kernel. It could be debated whether this is a more meaningful reference, but it provides a more consistent reference for the performance model and makes it possible to obtain better agreement between the model and measurement.

6.3 Parallel Performance Model Sensitivity

In this section the performance model is evaluated for a range of hypothetical hardware characteristics and OpenMP run time library overhead. The purpose of this analysis is to quantify the effect of the hardware or supplied OpenMP or MPI libraries on the algorithms performance, and to identify regimes in which more optimal performance may be achieved. Much like the model validation in the previous section, the model is analyzed separately for each type of decomposition.

First, the effect of the network latency and bandwidth on the spatial decomposition overhead is examined, specifically the T_{space} term of Eq. (5.12). In Figure 6.10 this function is plotted against the bandwidth and latency of the network hardware.

The measured range of latency and bandwidth for the test machine, Titan, is highlighted by the magenta rectangle.

As the bandwidth limits to large numbers and the latency limits to 0, T_{space} asymptotically approaches the reduced T_{BCUp} time. The contours of Figure 6.10 show that T_{space} approaches this asymptote rather quickly. Furthermore, very little reduction in the overhead is obtained once the bandwidth exceeds 10 MB/s and the latency is less than 10 µs. This suggests that efforts to improve the performance of the network architecture would have little benefit in reducing the overhead of the spatial decomposition, and that the performance of the network hardware on Titan is already near optimal for this algorithm.

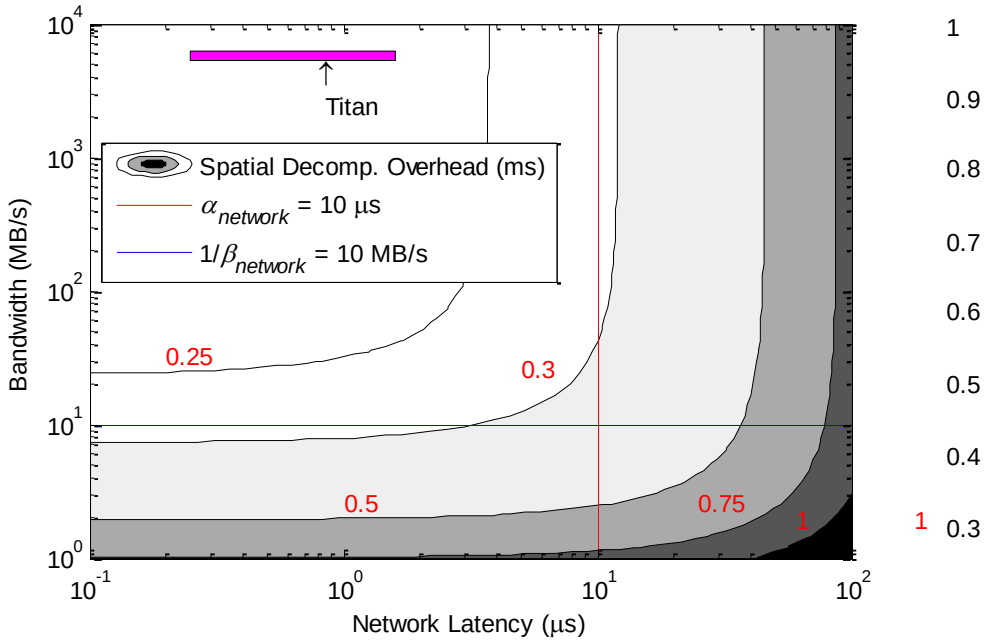


Figure 6.10 – Sensitivity of the spatial decomposition overhead to network hardware characteristics

In Figure 6.11, the competing factors of the "max" function of Eq. (5.18) are plotted along with the time to perform a sweep for a problem as a function of the number of spatial domains. This figure is therefore representative of the spatial strong scaling. The problem assumed in generating the data in this figure consists of a block of cubic pin cells that is 64x64x64, which represents roughly a 3x3 block of assemblies at roughly 1/5th their full height. Two main observations can be made about Figure 6.11. The first is that the time to do the spatial communication is always greater than the time to do any

remaining useful work when only one angular domain and one thread are assumed. The second observation is that T_{sweep} is still very close to ideal, even though the spatial decomposition overhead is the dominant term. This is because the overhead is still approximately an order of magnitude lower than the sweep time at the highest possible decomposition.

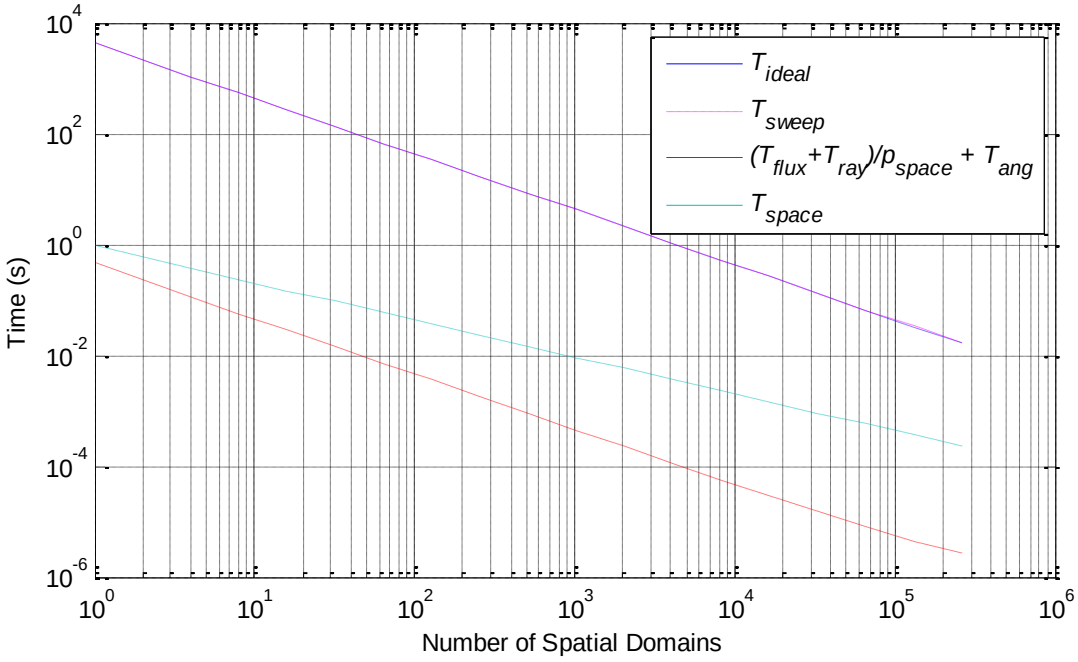


Figure 6.11 – Estimated execution times for spatial strong scaling

Next, for the angular decomposition, the sensitivity of the overhead is evaluated against the network hardware characteristics, and the problem size and number of domains. These sensitivities are examined for the three `MPI_Allreduce` algorithms given by Eq. (5.13), Eq. (5.14), and the semi-empirical model of Eq. (6.5) for the Cray MPI library on Titan. The sensitivities to the network hardware are shown in Figure 6.12 through Figure 6.14, and the measured network hardware latency and bandwidth are indicated by a magenta box in each figure. The Cray library and Rabensiefner's `MPI_Allreduce` algorithm clearly outperform the binary tree algorithm. The overhead is observed to be largely a function of the network bandwidth and, the magnitude of the overhead is about 1000x times higher for the angular decomposition compared to the spatial decomposition. Additionally, any increase in bandwidth above 100 MB/s provides little benefit to reducing the angle decomposition overhead.

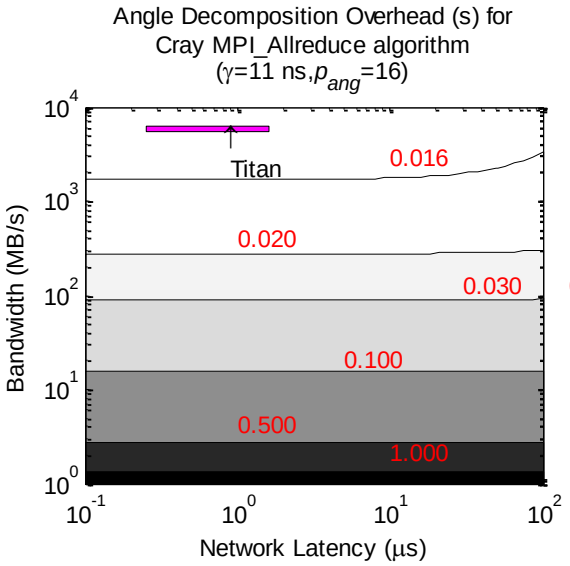


Figure 6.12 – Sensitivity of angle decomposition overhead to network hardware characteristics for Cray MPI_Allreduce algorithm

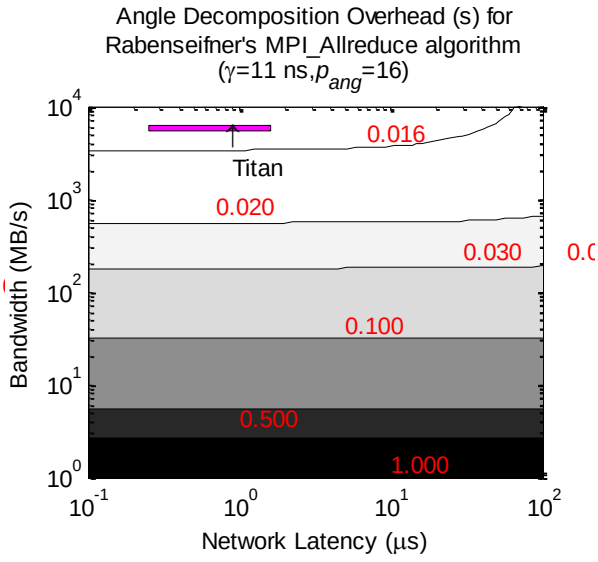


Figure 6.13 – Sensitivity of angle decomposition overhead to network hardware characteristics for Rabenseifner's MPI_Allreduce algorithm

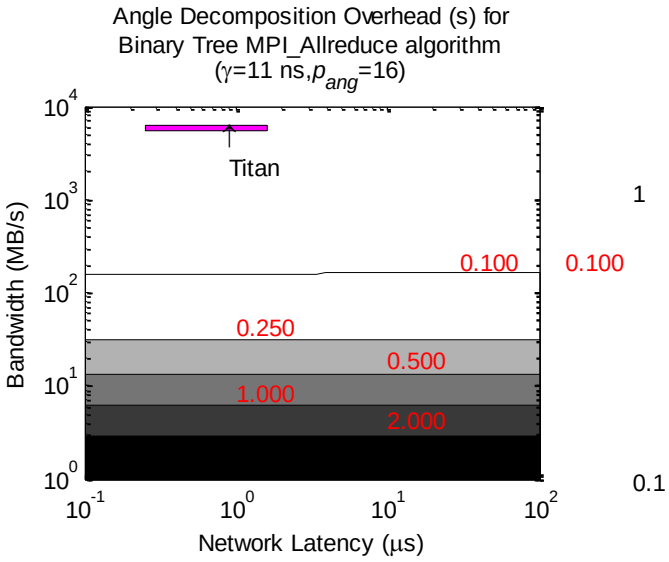


Figure 6.14 – Sensitivity of angle decomposition overhead to network hardware characteristics for binary tree MPI_Allreduce algorithm

In Figure 6.15 through Figure 6.17, the sensitivity of the angular decomposition overhead is shown as a function of the number of flat source regions in the domain and the number of angular domains. For this analysis the basic discretization of the default case test used to validate the performance model is assumed, and the number of flat source regions and angular domains is increased by a factors of 2 up to 16. Once again, the binary tree algorithm is clearly outperformed by the other two algorithms. Overall,

the angular decomposition overhead is fairly insensitive to the number of flat source regions once that number exceeds ~500,000. For the Cray and Rabenseifner all reduce algorithms, the overhead increases monotonically at a rate of approximately $O\left(\left(p_{angle}-1\right)/p_{angle}\right)$. In Figure 6.17, the curves have a staircase feature, which is due to the ceiling function in Eq. (5.13).

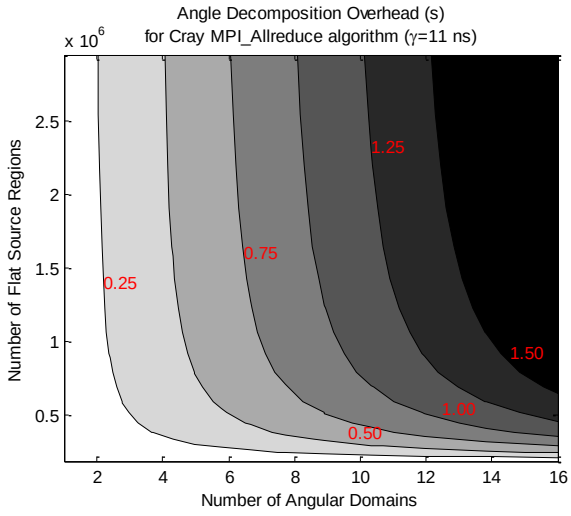


Figure 6.15 – Sensitivity of angle decomposition overhead to problem size and number of domains for Cray MPI_Allreduce algorithm

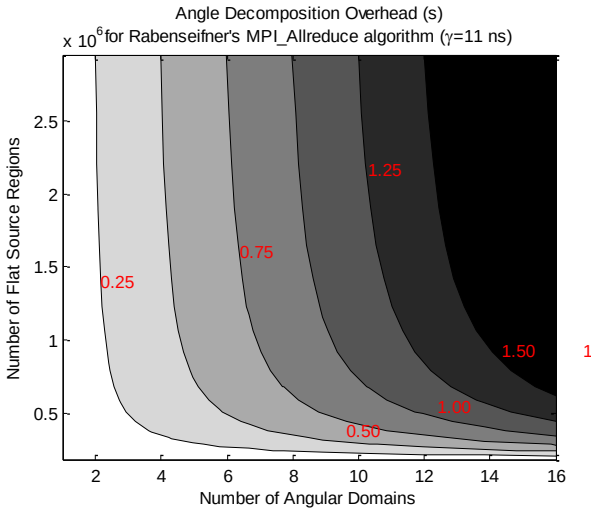


Figure 6.16 – Sensitivity of angle decomposition overhead to problem size and number of domains for Rabenseifner's MPI_Allreduce algorithm

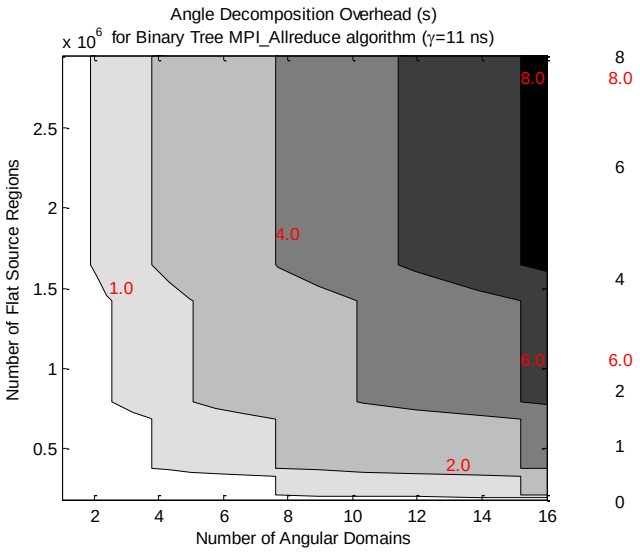


Figure 6.17 – Sensitivity of angle decomposition overhead to problem size and number of domains for binary tree MPI_Allreduce algorithm

The primary overhead for the ray decomposition comes from the OpenMP run time library. The parallel performance will not be especially sensitive to the serial hardware coefficients, such as t_f or α_j, but rather the ratio of these values to the run time library overhead. Therefore, the sensitivity of the ray decomposition overhead is evaluated as a function of the overhead values of the different functions used from the OpenMP run time library. Figure 6.18 shows the predicted strong scaling of the default case test problem, along with ideal scaling and the values of T_{ray} as the different OpenMP overheads limit to zero.

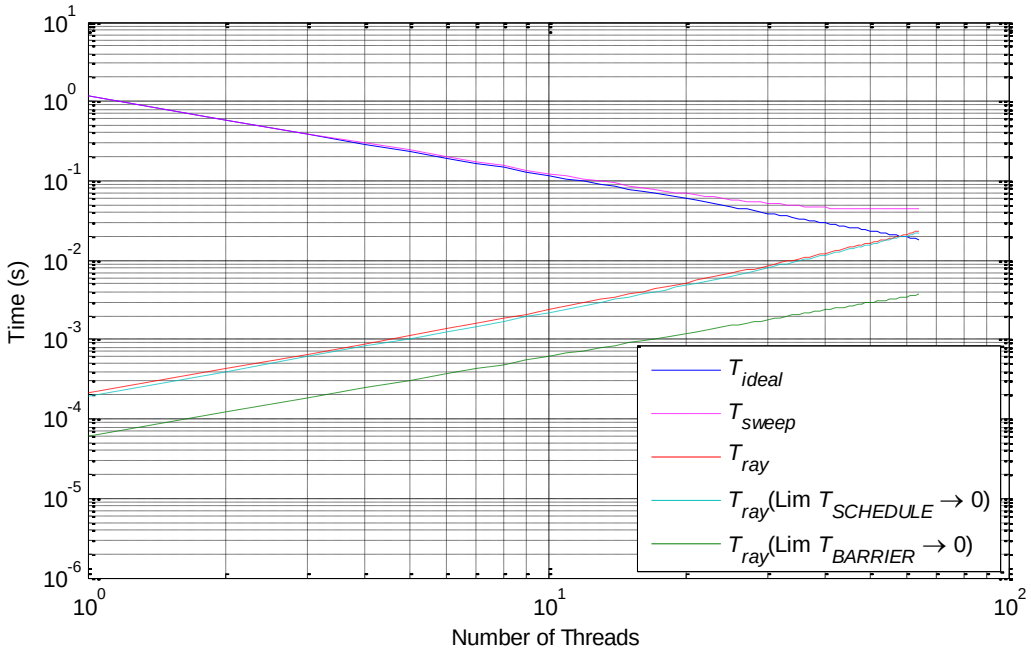


Figure 6.18 – Sensitivity of ray decomposition overhead to OpenMP library overhead

The sensitivity to the scheduling overhead is fairly small, which seems counterintuitive since the coefficient of this term in Eq. (5.20) is proportional to `nlongray`, which is typically quite large even for small problems. The scheduling overhead is small because the `chunk` size may be chosen to minimize the coefficient of $T_{SCHEDULE}$, thus minimizing its overhead. If optimization of the `chunk` size is performed, as it is represented by the model, then the ray scaling becomes relatively insensitive to the OpenMP scheduling overhead. Consequently, it is apparent in Figure 6.18, that the ray decomposition overhead is much more sensitive to the overhead of the barriers and

`single` construct. This indicates a potential future improvement to the parallel 3-D MOC kernel.

6.4 Decomposition Strategy for Optimizing Parallel Performance

The focus of this section is to understand the impact of changes to the parallel decomposition on the overall parallel efficiency of the kernel, and specifically to provide answers to the questions: If one has *n* processors, is there an optimal decomposition, and if so, what is it? For this study the basic hardware characteristics of Titan are used, and the assumed problem for the analysis is an idealized quarter core PWR that uses pin modular ray tracing and a pin-wise discretization that is consistent with the default case performance test problem studied in this chapter and Chapter 5.

First, in Figure 6.19, the strong scaling efficiency is shown for a single pin cell based on angle and ray decomposition. This plot shows that there is very little work for a problem of this size before the overhead of the various decompositions begins to dominate. The figure also shows that the ray decomposition scales with better efficiency than the angular decomposition, and perhaps only 4 to 8 processors can be used to efficiently parallelize the smallest spatial subdomain. The other observation of note for Figure 6.19 is that the rate at which the parallel efficiency decreases for the number of angles increases with the number of threads.

Figure 6.20 through Figure 6.24 show the predicted strong scaling efficiency for a quarter core PWR sized problem using pin modular ray tracing. With only one thread, the model predicts excellent scaling (>90%) out to $O(10^5)$ processors. By comparison, the change in strong scaling efficiency is relatively insensitive to the number of angular domains, meaning that for a single thread, the angular domain could be decomposed by a factor of almost 16 without a significant loss in efficiency.

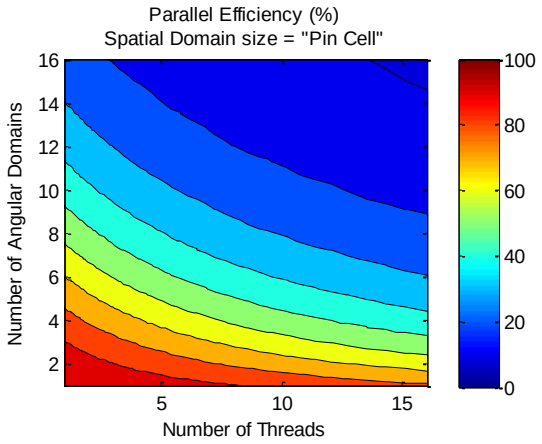


Figure 6.19 – Estimated angle-ray strong scaling efficiency for a pin cell

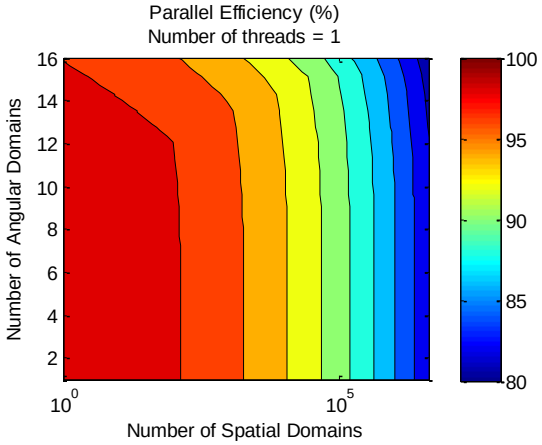


Figure 6.20 – Estimated space-angle strong scaling for a PWR 1/4 core (1 thread)

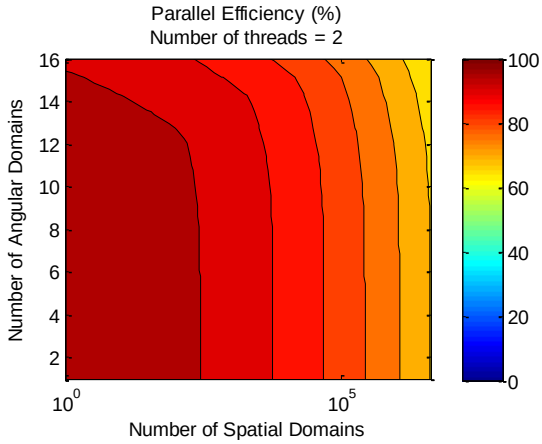


Figure 6.21 – Estimated space-angle strong scaling for a PWR 1/4 core (2 threads)

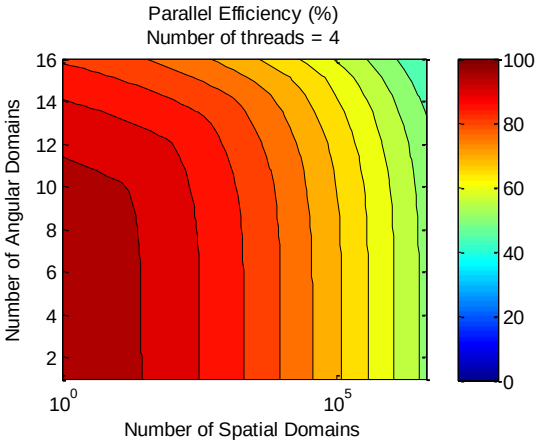


Figure 6.22 – Estimated space-angle strong scaling for a PWR 1/4 core (4 threads)

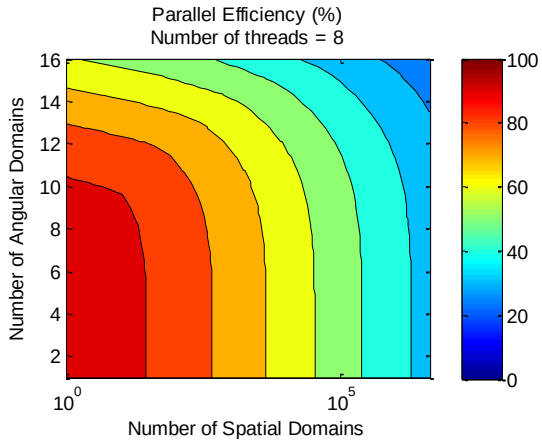


Figure 6.23 – Estimated space-angle strong scaling for a PWR 1/4 core (8 threads)

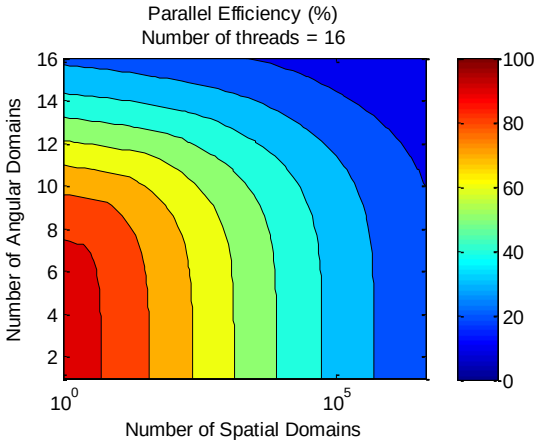


Figure 6.24 – Estimated space-angle strong scaling for a PWR 1/4 core (16 threads)

As the number of threads is increased, the strong scaling efficiency becomes noticeably worse. This is consistent with observations made from Figure 6.19, at which point the single pin cell problem becomes over decomposed. And as noted in Figure 6.19, the parallel efficiency degrades more rapidly for the angular decomposition when more threads are used. A similar effect is seen for the number of spatial domains. That is to be expected, since the amount of work in a given domain is decreased significantly by the number of threads. Figure 6.25 shows the aggregate data of Figure 6.20 through Figure 6.24 as a function of the number of processors. In Figure 6.25 it is observed that the model predicts that parallel efficiencies greater than 90% can be achieved out to nearly 30 million processors, using only space and angle decomposition. However, it is highly unlikely this performance could be observed in practice because the model assumes the hardware and run time system will scale to this many cores. The construction of computers having $O(10^6)$ processors still has yet to be demonstrated, and the current node architecture is likely to change drastically in the next generation of leadership class architectures.

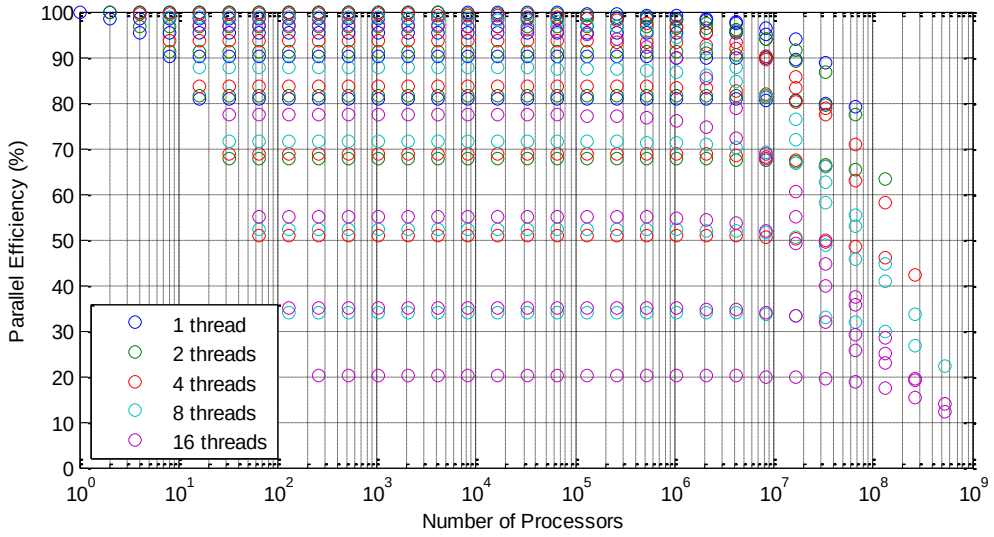


Figure 6.25 – Estimated strong scaling efficiency for PWR 1/4 core

Also in Figure 6.25, the rate of decrease in parallel efficiency is faster for space-ray decomposition than it is for space-angle decomposition. Therefore, in order to maximize the parallel efficiency it is essential to maximize the spatial decomposition. If more processors are available, then it is recommended to use the angular decomposition to add a factor of 2 to 8 more domains. However, if the spatial domain is not maximized,

perhaps because of poor load balancing or an insufficient number of processors for the next level of spatial decomposition, then it is better to add more processors through ray decomposition rather than angle decomposition. This analysis assumed perfect load balancing in the decomposition. In practice, this is not always possible, and using decompositions with a poor load balance will negatively affect the parallel efficiency much more than the estimate in this parametric study.

6.5 Summary

This chapter presented several new key insights about the parallelism of the 3-D MOC algorithm. First, new metrics for the parallel performance were introduced and defined. Then the execution time models derived in Chapter 5 were validated against measurements for the parallel execution. This required the use of several new micro-benchmarks for MPI and OpenMP to measure the network hardware's latency and bandwidth, as well as run time library overheads. It was found that having an accurate execution time model for `MPI_Allreduce` is essential to accurately predicting the angular decomposition overhead. The comparison of the parallel performance model to measurement showed that the model agreed quite well with measurement, with an absolute difference of less than 7%, in the prediction of the parallel efficiency. The other conclusion from this comparison was that special attention must be paid to the hardware used to obtain the measurement, and that when cores are added, the model assumes that this includes an additional floating point unit. In other words, the model assumes perfect scaling in the hardware. This may not always be the case with the hardware. However, the differences can be overcome if a more consistent reference measurement is used to account for the model's assumptions with respect to the hardware.

After establishing confidence in the parallel performance model, sensitivities to several factors in the model were examined through parametric studies. The overhead for the spatial and angular decomposition were examined separately with respect to the network's latency and bandwidth. Both types of decomposition were shown to be relatively insensitive to the network latency, provided the network bandwidth was at least 100 MB/s. Additionally, further increasing the network bandwidth beyond 100 MB/s or reducing the network latency below 0.1 ms would give little improvement in

performance. Since many modern high performance compute clusters already have networks with a higher bandwidth and lower latency, it may be concluded that these types of decompositions will continue to perform well on future network architectures.

The performance of the angular decomposition was also evaluated against the problem size and number of domains for various algorithms for `MPI_Allreduce`. This was shown to have a significant effect and that the preferred all reduce algorithms are those that are more optimal for large message sizes. While the current implementation of the parallel 3-D MOC kernel performs an all reduce, this may not strictly be required.

The overhead for the spatial decomposition was also evaluated against the number of domains, and it was shown that when other decompositions are not used, the overhead for the spatial decomposition is always greater than the amount of time it takes to do any remaining useful work. This is because the spatial decomposition is limited by the time required to copy the data to global memory. Also, the amount of data to be copied decreases at the rate at which the subdomain surface area is decreased, rather than by the subdomain's volume; that decreases faster for cuboids.

The ray decomposition overhead was shown to be limited mostly by the overhead of the synchronization of the threads which suggests a possible future area of improvement to the current implementation. This is assuming that the chunk size has been optimized to minimize the overhead of the OpenMP scheduling algorithm for the parallelized loop over long rays.

Finally, the parallel performance model for the strong scaling efficiency was evaluated for various decompositions, assuming problem size of the order of a quarter core PWR. The parallel performance model estimates that up to 30 million processors can be used while still maintaining efficiency greater than 90%, provided the hardware and run-time system scale. This analysis also showed that the best way to decompose a problem is to perform spatial decomposition to the point where the subdomain size is approximately a 2x2x2 or 4x4x4 block of pin cells. Then the number of ray or angle domains on each spatial subdomain can be increased by a factor of 8 and still have parallel efficiencies near 90%. Since the performance model assumes ideal load balancing for each decomposition, future work might focus on examining cases in which there is not an ideal load balance for some decomposition.

Chapter 7

ALGORITHM CONVERGENCE OPTIMIZATION WITH CMFD ACCELERATION

The primary focus of the thesis thus far has been on the optimization of the parallel performance of the 3-D MOC transport sweep. To some extent the optimal performance was achieved at the expense of the rate of convergence of the iterative solution scheme. This is illustrated by considering a problem that is purely absorbing with a known source. For such a problem, the transport sweep with one spatial domain would converge in one iteration, but for decomposed spatial domains, it would require more than one iteration. Therefore, this chapter focuses on further optimization of the algorithm by accelerating the convergence of the iterative scheme and reducing the overall number transport sweeps that must be performed to achieve a converged solution.

7.1 Convergence Acceleration

Considerable research has been performed on methods to accelerate iterative convergence of neutron transport problems [58], [59], [60], [61], [62]. In this section, the basic underlying concept of an acceleration technique is developed. The basic approach to acceleration has been to use what are known as *synthetic* acceleration techniques. The essential idea of synthetic acceleration can be summarized as follows. Suppose a problem has the following form:

$$(\mathbf{M}-\mathbf{F})\vec{\phi}=\vec{S}, \qquad \textbf{Eq. (7.1)}$$

where **M**-**F** is difficult to invert directly, but **M** is relatively easy to invert. Then an iterative scheme that should work well would be:

$$\vec{\phi}^{(\ell+1)}=\mathbf{A}\vec{\phi}^{(\ell)}+\mathbf{M}^{-1}\vec{S}, \quad \mathbf{A}=\mathbf{M}^{-1}\mathbf{F}. \qquad \textbf{Eq. (7.2)}$$

The rate of convergence of this iterative scheme will depend on the spectral radius of **A**. If **A** has a "large" spectral radius (less than, but close to unity) and is slow to converge, then it is possible to accelerate convergence using a new low-order operator **L**. First, one can show that the exact solution, $\vec{\phi}$, may be obtained from two consecutive iterates by subtracting the term (**M**-**F**)$\vec{\phi}^{(\ell+1)}$ from both sides of Eq. (7.1) and substituting Eq. (7.2) on the right hand side giving:

$$(\mathbf{M}-\mathbf{F})\left(\vec{\phi}-\vec{\phi}^{(\ell+1)}\right)=\mathbf{F}\left(\vec{\phi}^{(\ell+1)}-\vec{\phi}^{(\ell)}\right). \qquad \textbf{Eq. (7.3)}$$

Next, if one substitutes the low-order operator **L** for **M**-**F** in Eq. (7.3), where **L** is close to **M**-**F** and easily invertible, then one may use the new iterative scheme:

$$\vec{\phi}^{(\ell+1/2)}=\mathbf{A}\vec{\phi}^{(\ell)}+\mathbf{M}^{-1}\vec{S}, \qquad \textbf{Eq. (7.4)}$$

$$\vec{\phi}^{(\ell+1)}=\mathbf{L}^{-1}\mathbf{F}\left(\vec{\phi}^{(\ell+1/2)}-\vec{\phi}^{(\ell)}\right)+\vec{\phi}^{(\ell+1/2)}. \qquad \textbf{Eq. (7.5)}$$

Since **L** is close to **M**-**F** the iterate $\vec{\phi}^{(\ell+1)}$ of Eq. (7.5) should be close to the converged solution, $\vec{\phi}$, of Eq. (7.1). Eq. (7.5) is also assumed to be cheaper to evaluate than Eq. (7.4) and the operator $\mathbf{L}^{-1}\mathbf{F}$ should have better convergence properties than **A**. Very early on it was noted that diffusion-based operators were good low-order operators for synthetic acceleration of the transport equation [63], and significant research has gone into developing diffusion synthetic acceleration (DSA) techniques [58].

However, it should be noted in the above description of synthetic acceleration, Eq. (7.1) represents a fixed source problem and a linear update. Because the steady-state

problem of interest for a reactor is an eigenvalue problem, a non-linear acceleration technique would be more appropriate. One such method, the coarse mesh finite difference (CMFD) method, has been used with pronounced success for reactor analysis for the last several decades. CMFD was developed independently of many of the other diffusion synthetic acceleration schemes, but in recent work [64] it was shown that CMFD is algebraically equivalent to non-linear coarse mesh DSA. Therefore, CMFD was the logical choice for an acceleration technique in the research here and the derivation of this method is discussed in the following section.

7.2 Coarse Mesh Finite Difference

The *coarse mesh finite difference* method (CMFD) was originally developed as a technique for the nodal diffusion based methods used in reactor analysis [61]. However, its fundamental concept applies equally as well to transport methods, and has been shown to be very effective at accelerating 2-D MOC transport methods where it has been used extensively.

In CMFD, the lower order acceleration equation is based on the multi-group diffusion equation shown in Eq. (7.6). The discretized form of this equation is used to define the node balance and is shown in Eq. (7.7) where the subscript *s* is used to denote a surface of a node and *j* is a node index. $A_{j,s}$ is the area of surface *s* of node *j*, and *Vj* is the node volume. The over-bar notation is also added to indicate a node-averaged quantity. Node average quantities are defined in Eq. (7.10) and Eq. (7.11).

$$-\nabla\cdot\left(D_g(\vec{r})\nabla\phi_g(\vec{r})\right)+\Sigma_{t,g}(\vec{r})\phi_g(\vec{r})= \left[\sum_{g'=1}^{G}\left(\Sigma_{s0,g'\to g}(\vec{r})+\frac{\chi_g}{k_{eff}}\nu\Sigma_{f,g'}(\vec{r})\right)\phi_{g'}(\vec{r})\right], \qquad \textbf{Eq. (7.6)}$$

$$\sum_s \bar{J}^{net}_{j,g,s}A_{j,s}+\bar{\Sigma}_{t,j,g}\bar{\phi}_{j,g}V_j=V_j\left[\sum_{g'=1}^{G}\left(\Sigma_{s0,j,g'\to g}+\frac{\chi_g}{k_{eff}}\nu\Sigma_{f,j,g'}\right)\phi_{j,g'}\right]. \qquad \textbf{Eq. (7.7)}$$

In classic diffusion theory, the net current that is derived is based on Fick's Law with a finite difference approximation in space, and is shown in Eq. (7.8), where $h_{j,s}$ is the

distance between node *j* and its neighboring node on surface *s*. CMFD introduces a correction coefficient, $\hat{D}_{j,g,s}$, so that the expression for the net current is given by Eq. (7.9).

$$\bar{J}^{net,diff}_{j,g,s} = -\tilde{D}_{j,g,s}\left(\bar{\phi}_{j,g} - \bar{\phi}_{j,g,s}\right), \quad \tilde{D}_{j,g,s} = \frac{2\bar{D}_{j,g}\bar{D}_{j,g,s}}{h_{j,s}\left(\bar{D}_{j,g} + \bar{D}_{j,g,s}\right)}, \qquad \textbf{Eq. (7.8)}$$

$$\bar{J}^{net}_{j,g,s} = -\tilde{D}_{j,g,s}\left(\bar{\phi}_{j,g} - \bar{\phi}_{j,g,s}\right) + \hat{D}_{j,g,s}\left(\bar{\phi}_{j,g} + \bar{\phi}_{j,g,s}\right). \qquad \textbf{Eq. (7.9)}$$

CMFD also introduces the coarse mesh, or node, concept on which the diffusion equation is solved. This requires the development of restriction and prolongation transfer operators for the solution between the fine mesh defined by the MOC flat source region mesh, and the CMFD coarse mesh. The restriction operator, also known as homogenization, which reduces the fine mesh solution onto the course mesh and is shown in Eq. (7.10) and Eq. (7.11) for the cross sections and scalar flux, respectively. The prolongation operator is used to transfer the coarse mesh solution onto the fine mesh for the scalar flux and boundary condition are given by Eq. (7.12) and Eq. (7.13), respectively.

$$\bar{\Sigma}_{x,j,g} = \frac{\sum_{i \in j} \Sigma_{x,i,g}\phi_{i,g}V_i}{\sum_{i \in j} \phi_{i,g}V_i}, \qquad \textbf{Eq. (7.10)}$$

$$\bar{\phi}_{j,g} = \frac{\sum_{i \in j} \phi_{i,g}V_i}{\sum_{i \in j} \phi_{i,g}}, \qquad \textbf{Eq. (7.11)}$$

$$\phi_{i,g}^{l+1} = \phi_{i,g}^{l+1/2} \frac{\bar{\phi}_{j,g}^{l+1}}{\bar{\phi}_{j,g}^{l+1/2}} \quad i \in j, \qquad \textbf{Eq. (7.12)}$$

$$\varphi_{i,g,m,k}^{in,l+1} = \varphi_{i,g,m,k}^{in,l+1/2} \frac{\bar{J}_{j,g,s}^{in,l+1}}{\bar{J}_{j,g,s}^{in,l+1/2}} \quad k \in s, i \in j. \qquad \textbf{Eq. (7.13)}$$

The correction factor, $\hat{D}_{j,g,s}$, introduced in Eq. (7.9) is then computed as shown by Eq. (7.14), where the fine mesh transport method computes the neutron net current along the surfaces of the coarse mesh. Consequently this correction factor along with cross section homogenization creates equivalence between the solution of the fine mesh MOC equations and the coarse mesh diffusion equations. The homogenization process in general preserves all the node volume integrated quantities based on the fine mesh solution, and specifically the node average reaction rates. The correction factor of Eq. (7.9) and Eq. (7.14) allows the low order system to also preserve the node surface integrated quantities of the fine mesh solution, and specifically the average leakage. Because of this equivalence, the multiplication factor, k_{eff}, of the CMFD linear system is the same as that of the fine mesh transport method computed from source iteration when $\hat{D}_{j,g,s}$ is converged.

$$\hat{D}_{j,g,s} = \frac{\bar{J}_{j,g,s}^{net} + \tilde{D}_{j,g,s}\left(\bar{\phi}_{j,g} - \bar{\phi}_{j,g,s}\right)}{\left(\bar{\phi}_{j,g} + \bar{\phi}_{j,g,s}\right)}. \qquad \textbf{Eq. (7.14)}$$

The iterative solution algorithm with CMFD then becomes:

While not converged

1. Perform Transport Sweep (step 2 Figure 3.4):
   $\left\{\vec{\phi}_g^{(\ell+1/2)}, \vec{\varphi}_g^{in,(\ell+1/2)}\right\} \leftarrow f_{transport}\left(\vec{\varphi}_g^{in,(\ell)}, \vec{\phi}_g^{(\ell)}\right)$
2. Compute node averaged values for CMFD coefficients: Eq. (7.10), Eq. (7.11), and Eq. (7.14)
   $\left\{\overline{\phi}_{j,g}^{(\ell+1/2)}, \overline{\Sigma}_{x,j,g}, \hat{D}_{j,g,s}\right\} \leftarrow f_{\text{hom}}\left(\phi_{i,g}^{(\ell+1/2)}, \Sigma_{x,i,g}\right)$
3. Solve CMFD balance equation, Eq. (7.7) and Eq. (7.9), for node averaged scalar flux
   $\overline{\phi}_{j,g}^{(\ell+1)} \leftarrow f_{CMFD}\left(\overline{\phi}_{j,g}^{(\ell+1/2)}, \overline{\Sigma}_{x,j,g}, \hat{D}_{j,g,s}\right)$
4. Update fine mesh solution: Eq. (7.12) and Eq. (7.13)
   $\left\{\vec{\phi}_g^{(\ell+1)}, \vec{\varphi}_g^{in,(\ell+1)}\right\} \leftarrow f_{pro}\left(\vec{\overline{\phi}}_g^{(\ell+1)}, \vec{\overline{\phi}}_g^{(\ell+1/2)}\right)$
5. Update fission source and k_{eff}
6. Check if solution is converged

Figure 7.1 – Solution algorithm with CMFD

7.2.1 Spatial Domain Decomposed Coarse Mesh Finite Difference

Recently other work has been done to extend the CMFD theory to the application of spatially decomposed transport problems [65] that can be solved in parallel. In that work, additional update equations for the parallel subdomain interface angular fluxes were derived and the convergence properties of the system were examined using a Fourier analysis of a model problem. Numerical results were produced in 1-D for a discrete ordinates transport method that agreed with the convergence rate predicted by the Fourier analysis.

The spatially domain decomposed CMFD (SDD-CMFD) method was implemented in the work here to accelerate the parallel 3-D MOC kernel described in Chapter 4. In the original work the equation to update the interface angular fluxes between parallel domains is derived from P_1 theory and is given by Eq. (7.15). This essentially replaces Eq. (7.13), although in [65] the authors note that other sensible update equations, such as Eq. (7.13), may be appropriate.

$$\varphi_{i,g,m,k}^{in,(\ell+1)} = \varphi_{g,m,k}^{in,(\ell+1/2)} \left(\frac{\bar{\phi}_{j,g,s}^{(\ell+1)} + 3\left(\vec{\Omega}_m \cdot \vec{n}_s\right)\bar{J}_{j,g,s}^{net,(\ell+1)}}{\bar{\phi}_{j,g,s}^{(\ell+1/2)} + 3\left(\vec{\Omega}_m \cdot \vec{n}_s\right)\bar{J}_{j,g,s}^{net,(\ell+1/2)}} \right) \quad k \in s, i \in j. \qquad \textbf{Eq. (7.15)}$$

The convergence behavior predicted by Fourier analysis for the SDD-CMFD method using Eq. (7.15) is reproduced from [65] and shown in Figure 7.2 as a function of the scattering ratio and optical thickness.

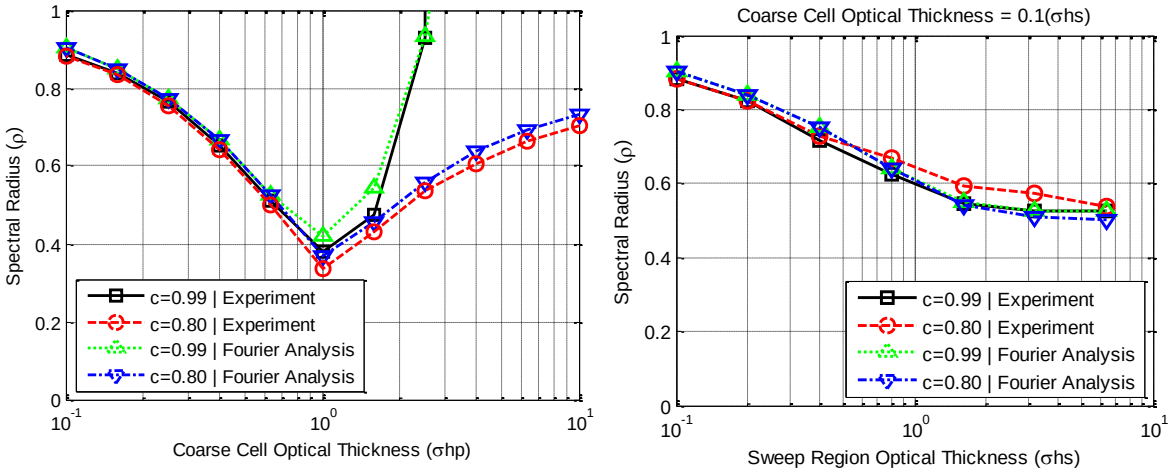


Figure 7.2 – Convergence properties of SDD-CMFD

7.3 SDD-CMFD Convergence with Parallel 3-D MOC

In this section, the implementation of the SDD-CMFD is evaluated for multi-dimensional problems and two update equations by comparing the spectral radius of several cases to that predicted by theory. First, simple infinite homogeneous media cases are decomposed along one dimension to reproduce the convergence behavior from previous work. Then these cases are extended to decompose the problem in two dimensions and then all three dimensions. The convergence of these other cases are then compared to the convergence predicted by Fourier analysis and the results are discussed.

7.3.1 Solution of SDD-CMFD Equations

The SDD-CMFD equations in 3-D form a seven-stripe sparse linear system. This linear system is solved using the well known generalized minimum residual (GMRES) [66] Krylov method. The PETSc scientific software library [18] is used to provide the parallel

GMRES solver and a block ILU(0) preconditioner, since this library is already well optimized and has been shown to scale well on large parallel systems [21].

In this work the full space-energy system of equations is formed and solved simultaneously. Furthermore, the SDD-CMFD equations are used to compute the eigenvalue of the reactor system, since it will be equivalent to the eigenvalue of the computed by the algorithm of Figure 2.3 at convergence. The solution algorithm for the eigenvalue problem using the SDD-CMFD equations is essentially the power method and shown in Figure 7.1. However, because the full space-energy linear system is formed in this implementation, the convergence for the power method is further improved using Weilandt acceleration [67], or eigenvalue deflation.

The inclusion of SDD-CMFD acceleration technique into the overall algorithm for solving the reactor criticality problem introduces a great deal of complexity into quantifying the computational performance. Since the SDD-CMFD equations are less computationally expensive to evaluate and because PETSc is used to provide the solver for the SDD-CMFD equations, it is assumed that the time spent in this part of the calculation will be less than the time spent in the transport sweep that is the main focus of this research. Thus, a detailed performance model for the SDD-CMFD method and accelerated solution algorithm was not developed, and instead left for future work.

7.3.2 Model Problem Description

The original work [65] examined the convergence properties of SDD-CMFD as a function of several parameters, including the coarse cell optical thickness, the spatial subdomain optical thickness, the scattering ratio, the number of coarse cells per spatial subdomain and the number of fine cells per coarse cell. In this analysis, only two parameters are evaluated: the spatial subdomain optical thickness for a fixed coarse cell optical thickness and fixed number of fine cells per coarse cell, and the coarse cell optical thickness for a fixed number of coarse cells per spatial subdomain and fixed number of fine cells per coarse cell. Both cases use 1-group cross sections with a fixed scattering ratio, c, of 0.99, since higher scattering ratios are generally more limiting to the convergence of CMFD. The update equations for the boundary angular flux that are

examined are Eq. (7.15), which is a P_1 update, and Eq. (7.13) which is a double-P_0 (DP_0) update.

The model problem uses a fixed multiplication factor of 1.0, and during the iterations the spatial distribution of the fission and scattering source and spatial and angular boundary conditions are allowed to vary. The initial guess of the scalar flux and boundary angular flux is chosen to be random.

The spectral radius is determined by computing the error norm, ε, of Eq. (7.16) at each iteration. The spectral radius, ρ, is then related to this quantity as shown by Eq. (7.17).

$$\varepsilon = \sqrt{\left(\left\|\vec{\phi}^{(\ell)} - \vec{\phi}^{(\ell-1)}\right\|_2\right)^2 + \left(\left\|\vec{\varphi}^{in,(\ell)} - \vec{\varphi}^{in,(\ell-1)}\right\|_2\right)^2}, \qquad \textbf{Eq. (7.16)}$$

$$\varepsilon^{(\ell)} \approx \varepsilon^{(0)} \rho^{\ell}. \qquad \textbf{Eq. (7.17)}$$

A linear regression of the natural logarithm of this error norm as a function of the iteration number (Eq. (7.18)) is performed for a subset of the iterations for which the convergence behavior is smooth. The slope of the error norm can then be used to determine the spectral radius, as shown by Eq. (7.20). This is consistent with the methods used in the original work [55] for estimating the spectral radius by experiment.

$$\ln\left(\varepsilon^{(\ell)}\right) = a_1 \ell + a_0 \Rightarrow \varepsilon^{(\ell)} \approx a_0 \exp\left(a_1 \ell\right), \qquad \textbf{Eq. (7.18)}$$

$$\varepsilon^{(0)} = \exp\left(a_0\right), \qquad \textbf{Eq. (7.19)}$$

$$\rho = \exp\left(a_1\right). \qquad \textbf{Eq. (7.20)}$$

7.3.3 Results and Discussion

Figure 7.3 and Figure 7.4 show the comparison of the spectral radius predicted by the Fourier analysis from [65] and the experimentally computed spectral radii for the various update equations and spatial dimensionality. The 3-D MOC kernel is used in all cases to

produce the results from which the spectral radius is calculated. For the data points denoted as "1-D" in the legend, the problem is only discretized in 1-D, similarly for the "2-D" and "3-D" data points. In this way, the solution is able to behave as if the problem were 1-D or 2-D even though a 3-D transport method is being used.

Figure 7.3 shows the spectral radius as a function of the coarse cell optical thickness. The spatial subdomain consists of a single coarse cell, and the coarse cells have two fine mesh regions in 1-D. For 2-D, the number of fine cells per coarse cell is four; essentially replicating the 1-D discretization in the second spatial dimension. Similarly, for the 3-D case it is eight fine cells per coarse cell.

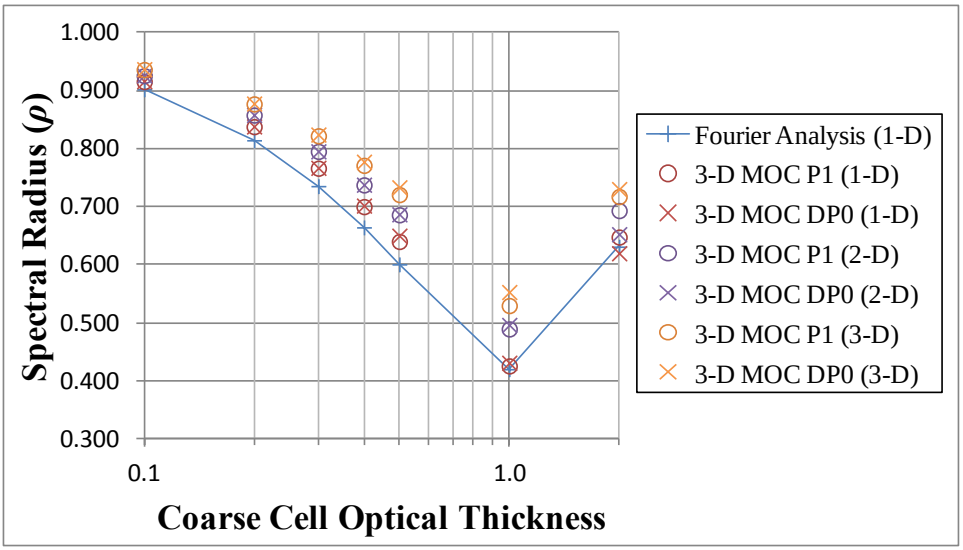


Figure 7.3 – Convergence properties of SDD-CMFD for c=0.99 and one coarse cell per spatial subdomain

It is observed that the 1-D results agree reasonably well with the Fourier Analysis, thus supporting the results of the previous work. In examining the effect of the spatial dimensionality it is observed that the spectral radius increases with increasing dimensionality. It is hypothesized that the reason the spectral radius increases with the spatial dimensionality is because the incoming angular flux on orthogonal surfaces of the spatial subdomain boundaries will always be coupled. This is due to the characteristic rays that are very close to the corners of the spatial subdomain boundaries. The optical thicknesses along these trajectories will always be less than in the 1-D case, where the optical thickness is based on the distance between opposing faces on the subdomain. It is

expected that this effect will reduce the rate of convergence. From the results in Figure 7.3 it would appear that this effect is more severe for smaller spectral radii. However, the absolute change in the magnitude of the spectral radius is approximately 0.1 between 1-D and 3-D, with an optical thickness of 1.0, or roughly 25% increase in the spectral radius from 1-D to 3-D. This suggests the SDD-CMFD method, while less effective in higher spatial dimensions, is still fairly effective at accelerating the convergence even in 3-D. The DP_0 update and P_1 update also agree well in 1-D and 2-D, but for some data points the agreement in 3-D is not as good. The possible reason for this is discussed later in this section.

Figure 7.4 shows the variation of the spectral radius as a function of the spatial subdomain's, or *sweep region*'s, optical thickness for a fixed coarse cell optical thickness of 0.1. The spatial domain optical thickness is varied by changing the number of coarse cells in the spatial subdomain. The number of fine cells per coarse cell vary in the same fashion in Figure 7.4 as they did for the results shown in Figure 7.3.

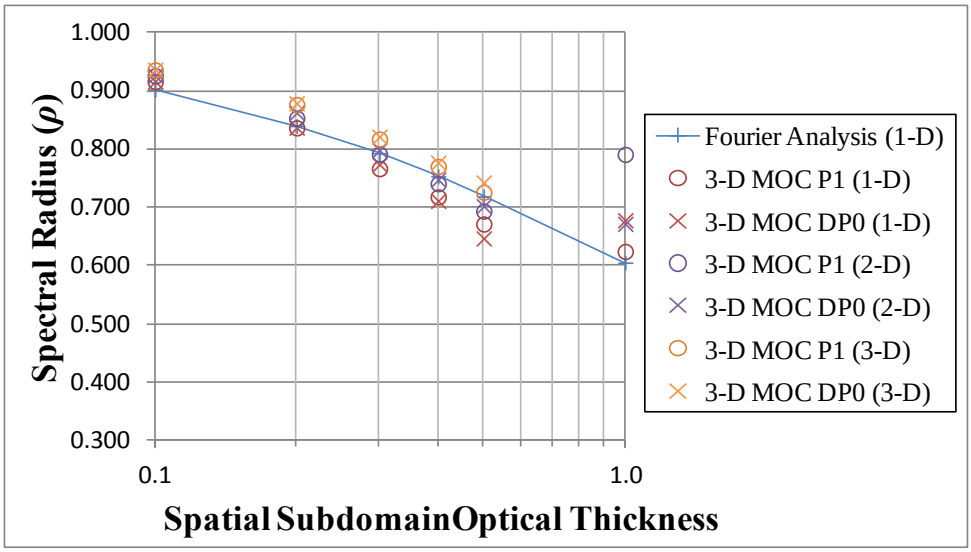


Figure 7.4 – Convergence properties of SDD-CMFD for c=0.99 and coarse cell optical thickness of 0.1

In Figure 7.4 the trend of the numerical data follows the general trend of the spectral radius predicted by the Fourier analysis, although there are some notable differences. The experimentally determined spectral radii are observed to decrease at a faster rate than that predicted by the Fourier analysis with an increasing spatial

subdomain optical thickness. The next notable difference is the spectral radius predicted for the 2-D case with the sweep region optical thickness equal to 1.0. In the 2-D case the spectral radius increases dramatically, rather than continuing to decrease. This is also assumed to be an effect of boundary conditions being coupled on orthogonal faces and from the neutron trajectories with smaller mean free paths that pass near the corners of the spatial subdomains. The data for the 3-D case with this optical thickness was not calculated because of initialization times that exceeded several hours. However, it is assumed the 3-D case would have been similar to the 2-D case. Finally, in comparing the experimentally determined spectral radii, the same trend of an increasing spectral radius with increasing spatial dimensionality is again observed.

The final result to be discussed further highlights the differences between the DP_0 and P_1 updates of the incoming angular flux boundary conditions. The previous figures showed reasonable agreement for either update method in most cases. However, in Figure 7.5 below, the full error history for the 2-D case with a spatial subdomain optical thickness of 1.0 with the P_1 and DP_0 update is shown to have a markedly different behavior after a certain number of iterations.

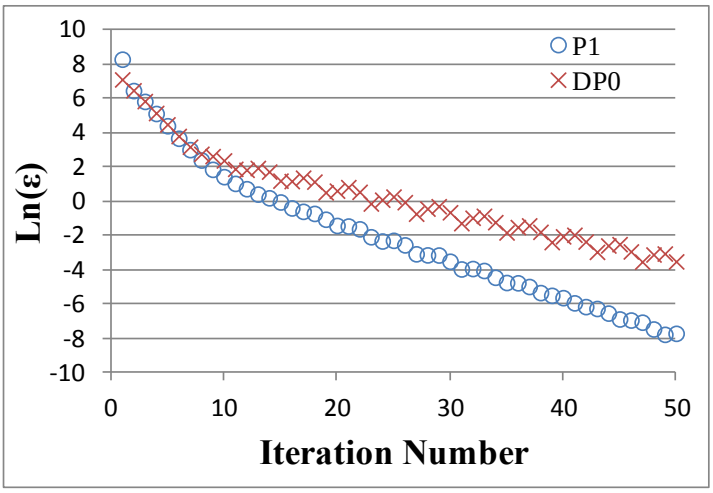


Figure 7.5 – Comparison of convergence properties of SDD-CMFD with DP_0 and P_1 updates of the boundary angular flux

In Figure 7.5 it is observed that the error history when using the DP_0 update becomes oscillatory and converges at a much slower rate after about 10 iterations. It is very likely that this occurs because another error mode in the problem becomes dominant

after some number of iterations. Because the DP_0 updates the boundary angular flux isotropocally, it is hypothesized that this dominant error mode must be an error mode that is related to the linear variation of the solution in angle. However, this effect is also observed in the 1-D numerical results, and so it may be a result of inconsistencies in the numerical iteration technique actually being 3-D instead of truly 1-D. To verify whether the source of the DP_0 convergence behavior is because of an error mode that is linearly dependent in angle or inconsistencies in the numerical experiment, a Fourier analysis using the DP_0 update equation will be performed in future work.

7.4 Summary

This chapter introduced the basic idea behind synthetic acceleration techniques. It then presented recent work on the application of one such synthetic acceleration technique, CMFD, to problems that are decomposed in space. In this previous work [65], the convergence properties of this method, SDD-CMFD, were predicted for model problems in 1-D by Fourier analysis and verified numerically with a 1-D method. The work of this chapter added to these experimental results by using a 3-D transport method on a similar model problem that was first designed to have a strictly 1-D solution. The numerical estimates of the spectral radius for the "1-D" problem agreed reasonably well with the previous results suggesting that the numerical experimentation method is sound. Numerical estimates of the convergence behavior of SDD-CMFD were then generated for 2-D and 3-D model problems, and looked at an alternative update equation that was based on a DP_0 approximation to the boundary angular flux, rather than a P_1 approximation. In these new results it was observed that the SDD-CMFD method was not as effective for multi-dimensional problems compared to 1-D, but showed a considerable improvement on the rate of convergence. Additionally, the DP_0 update was observed to produce convergence behavior similar to the P_1 update of the boundary angular fluxes in some cases. However, in others it was observed to be less efficient at accelerating the convergence.

Chapter 8

SOLUTIONS TO NUMERICAL BENCHMARKS

In this chapter the parallel performance of the 3-D MOC kernel is assessed using several numerical benchmarks. The purpose of doing this is not only to evaluate the accuracy and efficiency of the parallel 3-D MOC method, but also to examine the sensitivity of the accuracy and execution times to the level of discretization. In previous work [32], [33], the 3-D MOC method was used to compute solutions to one of the Takeda benchmarks [68]. This will be the first benchmark discussed in this chapter. The other benchmarks analyzed include the C5G7 benchmarks [69], [70] for a heterogeneous reactor model and a single PWR assembly based on one of the CASL AMA Benchmarks [71], that includes a more realistic PWR geometry.

8.1 Takeda Benchmark: Model 1

The Takeda Benchmark suite [68] is a set of 3-D transport benchmarks that include several simplified reactor core models for the purposes assessing 3-D transport codes for reactor applications. Only the first model is examined here, since it is intended to be representative of a small LWR core.

8.1.1 Model Description and Calculation Details

The model includes two cases for a rodded and unrodded condition. The benchmark specification includes four types of materials with two-group cross sections. The core model geometry is shown in Figure 8.1. In discretizing this model, modular ray tracing for a 1cm x 1cm x 1cm domain is used for all computed results reported. The convergence criteria used for all the calculations was $\varepsilon_{keff} < 10^{-6}$ and $\varepsilon_{flux} < 10^{-6}$, where

the eigenvalue residual, ε_{keff}, and the flux residual, ε_{flux}, are given by Eq. (8.1) and Eq. (8.2) respectively. All calculations were performed on Titan [54].

$$\varepsilon_{keff} \equiv \left| k_{eff}^{(\ell)} - k_{eff}^{(\ell-1)} \right|, \qquad \textbf{Eq. (8.1)}$$

$$\varepsilon_{flux} \equiv \left\| \vec{\phi}^{(\ell)} - \vec{\phi}^{(\ell-1)} \right\|_2. \qquad \textbf{Eq. (8.2)}$$

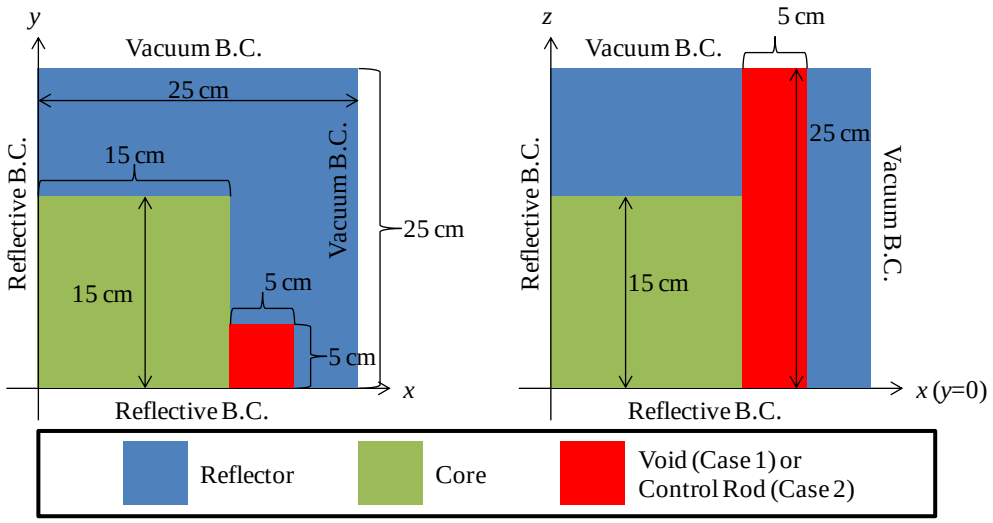


Figure 8.1 – Takeda benchmark problem 1 geometry

In calculating solutions to this benchmark, several discretizations were used to demonstrate mesh convergence and to show that the computed solution approaches the benchmark reference. The calculations were performed using several different parallel decompositions, and with and without SDD-CMFD (referred to as CMFD in the remainder of the chapter) for convergence acceleration. The different discretizations of the different phase spaces of the solution that were used are shown in Table 8.1.

Table 8.1 – Takeda problem 1 discretizations

Discretization Level	Flat Source Region Size	Angular Quadrature	Max Ray Spacing
Coarse	1 cm x 1 cm x 1 cm	S_4 Level Symmetric	0.1 cm
Medium	0.5 cm x 0.5 cm x 0.5 cm	S_8 Level Symmetric	0.05 cm
Fine	0.25 cm x 0.25 cm x 0.25 cm	S_{16} Level Symmetric	0.01 cm

For the medium discretization level (e.g. 0.5 cm sized flat source region, S_8 angular quadrature, and 0.05 cm ray spacing), the time to convergence for various parallel decompositions is also examined. In these calculations a basic source iteration is used instead, and the CMFD acceleration is not performed. This is to highlight the effect of the spatial decomposition on the rate of convergence. Finally, using this same discretization and a varying number of spatial domains, the effectiveness of the CMFD is examined.

8.1.2 Results and Discussion

Figure 8.2 shows the computed eigenvalues for the different discretizations, compared to the benchmark reference. In these figures it is observed that the eigenvalue change is relatively small with variations to the angular quadrature order and ray spacing, once the flat source region mesh is refined to $(0.5\text{ cm})^3$. This suggests that this mesh is sufficient for fully resolving the region-wise flat source. It is also observed that increasing the angular order has the next largest effect on k_{eff}. There is a more noticeable improvement in the solution going from S_8 to S_{16} than from S_4 to S_8. The ray spacing seems to have the smallest effect, which is likely due to the problem having little spatial heterogeneity, so even coarse ray discretizations numerically integrate the spatial mesh regions well. Using $(0.5\text{ cm})^3$ flat source regions with the S_8 quadrature and 0.05 cm ray spacing for the discretization give a solution that is reasonably accurate.

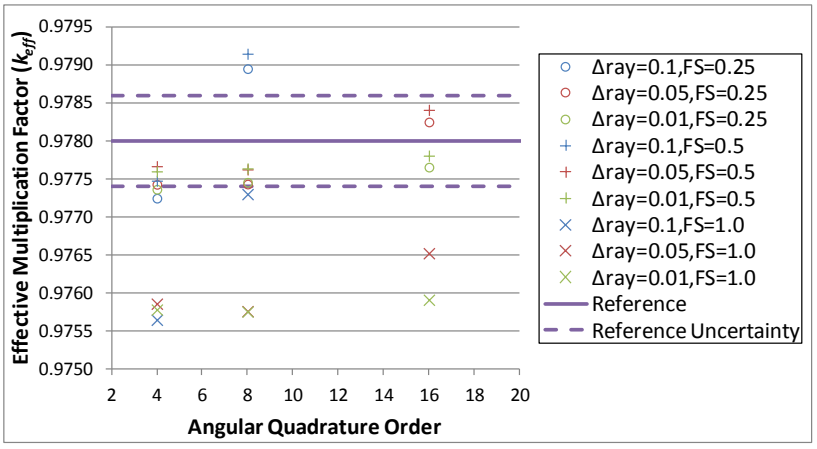


Figure 8.2 – Computed k_{eff} of Takeda benchmark for various discretizations

The effective multiplication factors for each benchmark case with the aforementioned discretization are shown in Table 8.2, along with the averages of the original benchmark participants' results and the reference eigenvalue. The region average fluxes for case 1 (unrodded) and case 2 (rodded) are shown in Table 8.3 and Table 8.4, respectively.

Table 8.2 – Comparison of reference average k_{eff} for Takeda benchmark model 1

Method	Case 1 k_{eff}	Case 2 k_{eff}	Case 1 Δk_{eff} (pcm)	Case 2 Δk_{eff} (pcm)
Reference	0.9780 (±0.0006)	0.9624 (±0.0006)	---	---
P_N	0.9778	0.9625	-20	10
S_4	0.9766	0.9630	-140	60
S_8	0.9766	0.9622	-140	-20
MPACT(3-D MOC)	0.97763	0.96253	-37	-13

Table 8.3 – Comparison of region average fluxes for Takeda benchmark model 1 case 1

Method		Core		Reflector		Void	
		Avg.	Rel. Diff.	Avg.	Rel. Diff.	Avg.	Rel. Diff.
Reference	1	4.7509E-03	0.10%	5.9251E-04	0.21%	1.4500E-03	0.47%
	2	8.6998E-04	0.12%	9.1404E-04	0.23%	9.7406E-04	0.63%
Monte Carlo	1	4.7835E-03	0.69%	5.9722E-04	0.79%	1.4529E-03	0.20%
	2	8.7841E-04	0.97%	9.2036E-04	0.69%	9.7684E-04	0.29%
P_N	1	4.7472E-03	-0.08%	5.9439E-04	0.32%	1.4096E-03	-2.79%
	2	8.6452E-04	-0.63%	9.2059E-04	0.72%	9.0113E-04	-7.49%
S_n	1	4.7650E-03	0.30%	5.9361E-04	0.19%	1.4453E-03	-0.32%
	2	8.7162E-04	0.19%	9.1520E-04	0.13%	9.6997E-04	-0.42%
MPACT (3-D MOC)	1	4.7445E-03	-0.13%	5.9552E-04	0.51%	1.4568E-03	0.47%
	2	8.7239E-04	0.28%	8.8513E-04	-3.16%	9.5427E-04	-2.03%

Table 8.4 – Comparison of region average fluxes for Takeda benchmark model 1 case 2

Method		Core		Reflector		CR	
		Avg.	Rel. Diff.	Avg.	Rel. Diff.	Avg.	Rel. Diff.
Reference	1	4.9125E-03	0.10%	5.9109E-04	0.21%	1.2247E-03	0.48%
	2	8.6921E-04	0.13%	8.7897E-04	0.23%	2.4604E-04	0.72%
Monte Carlo	1	4.9006E-03	-0.24%	5.8989E-04	-0.20%	1.2264E-03	0.14%
	2	8.6814E-04	-0.12%	8.8012E-04	0.13%	2.4615E-04	0.04%
P_N	1	4.8581E-03	-1.11%	5.8854E-04	-0.43%	1.1996E-03	-2.05%
	2	8.6003E-04	-1.06%	8.8412E-04	0.59%	2.4257E-04	-1.41%
S_n	1	4.8968E-03	-0.32%	5.8980E-04	-0.22%	1.2218E-03	-0.24%
	2	8.6751E-04	-0.20%	8.8074E-04	0.20%	2.4538E-04	-0.27%
MPACT (3-D MOC)	1	4.8785E-03	-0.69%	5.9171E-04	0.11%	1.2285E-03	0.31%
	2	8.6820E-04	-0.12%	8.5024E-04	-3.27%	2.5147E-04	2.21%

The eigenvalues compare within the statistical uncertainty of the reference and except for the group 2 fluxes in the reflector and void or core regions, the agreement of the region average fluxes with reference is comparable to the averages of the other participants. The cause of flux differences was not investigated rigorously, but is thought to be related to the discretization. Therefore, from these results it can be concluded that the methodology of the parallel 3-D MOC kernel is correct.

Figure 8.3 and Figure 8.4 show the parallel efficiency and speedup as defined by Eq. (6.1) and Eq. (6.2), respectively, for various decompositions measured against a reference calculation using a full NUMA node. For this problem, effective strong scaling (parallel efficiency > 80%) is achieved for up to 2048 processors using 64 spatial domains, 8 angular domains and 4 threads; only two other cases were run with processor counts larger than 2048. The first was with 4000 processors that used 125 spatial domains, 8 angular domains and 4 threads and had a parallel efficiency of 77%. The other case used 15625 cores to fully decompose the spatial domain and the parallel efficiency of this decomposition was 61%. The performance of the worst case, with 256 processors and 50% parallel efficiency used 1 spatial domain, 16 angular domains and 16 threads and is corroborated by the analysis of the performance model in Section 6.4. The average time for a transport sweep and the total solution time are also shown in Figure 8.5, where the "ideal" time is execution time on one processor divided by the *n* processors.

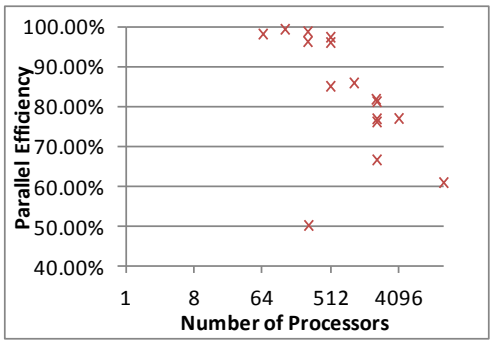


Figure 8.3 – Strong scaling parallel efficiency for Takeda problem on Titan

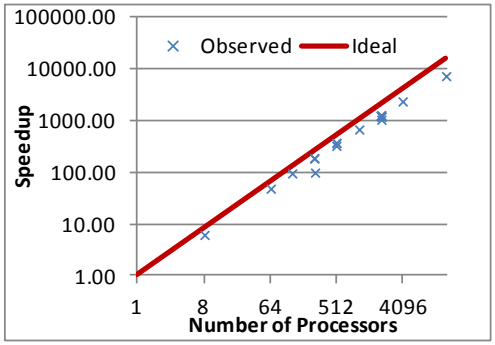


Figure 8.4 – Strong scaling speedup for Takeda problem on Titan

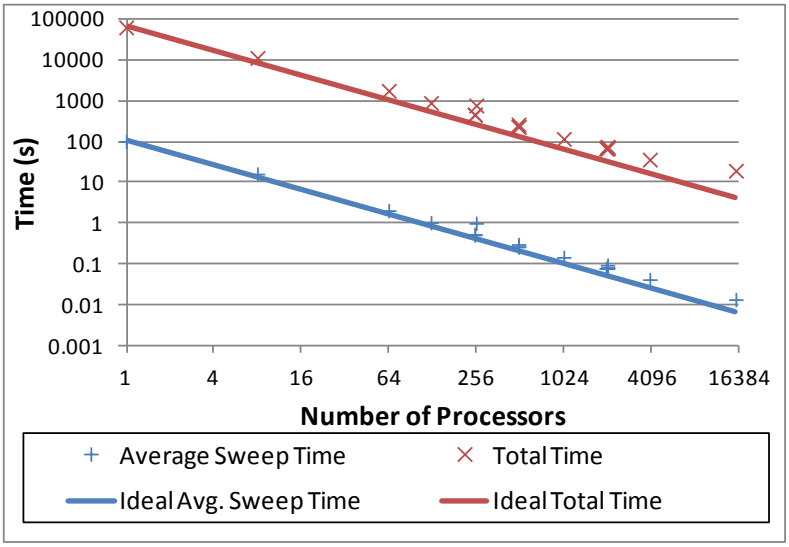


Figure 8.5 – Takeda problem run times with 3-D MOC for various parallel decompositions on Titan

The number of iterations and overall run time to converge for various decompositions with and without CMFD are shown in Table 8.5. From this data it is observed that the number of iterations to converge is far less sensitive to the spatial decomposition when CMFD is used. The factor by which the iterations are reduced ranges between 13 and 18 for this problem. The overall speedup achieved with CMFD does not scale exactly with the reduction in the number of iterations. This is to be expected, since CMFD introduces some computational overhead. To a point, the effective speedup increases with an increasing number of spatial domains. The effective speedup likely varies because the factor by which the number of iterations is reduced is changing. Additionally, this speedup is affected by the parallel efficiency of the CMFD linear solver, which is likely more important than the reason noted above. With 1000 spatial domains, the domain sizes vary between 2x2x2 and 3x3x3 blocks of CMFD nodes. The decrease in effective speedup from 125 domains to 1000 domains is most likely because the subdomain sizes are not balanced, and because subdomains are too small to solve the CMFD linear system efficiently in parallel. This highlights the importance of having a good parallel linear solver for CMFD.

Table 8.5 – Effectiveness of CMFD with spatial decomposition

Number of Spatial Domains	No CMFD		With CMFD		Effective Speedup
	No. of Iters.	Run Time (s)	No. of Iters.	Run Time (s)	
1	107	101224.00	8	10025	10.10
8	109	18759.70	7	1762	10.65
64	121	2985.12	7	240.21	12.43
125	124	1150.74	7	93.06	12.37
1000	146	303.15	11	39.86	7.61

The breakdown of the calculation time for MOC and the different components of the SDD-CMFD calculation is shown in Figure 8.7 through Figure 8.10. It is apparent in these figures that the MOC time dominates in all cases. However, the fraction of time spent in the CMFD portion of the calculation increases with the increasing number of spatial domains. This suggests that the scaling of the GMRES solver in PETSc does not perform as well as the 3-D MOC kernel. Therefore, fully decomposing a problem in space may not be optimal when using CMFD, especially if the spatial subdomain is a single CMFD node. The other interesting observation from the data in these figures is that the time to update the MOC solution takes almost as much or more time than it does to solve the CMFD equations. Although, the update component of the CMFD takes less time than the solve when the problem becomes over-decomposed.

The times for the MOC calculation, total CMFD calculation time, and solution time are shown in Table 8.6. There is a notable increase in the CMFD solution time between the 125 processor and 1000 processor case. It is possible that this may be improved by developing a solver more specific to the CMFD and parallel decomposition, but it appears from this data that the strong scaling efficiency of the CMFD solver decreases when the spatial subdomain sizes become smaller than ~200 CMFD nodes.

Table 8.6 – Scaling of calculation component times for Takeda benchmark

Number of Processors	Number of Iters.	Run Time (s)	Average Time (s)	
			MOC	CMFD
1	8	10025.00	9833.00	191.38
8	7	1762.37	1720.59	25.16
64	7	240.21	218.38	4.74
125	7	93.06	88.05	3.95
1000	11	39.86	29.89	8.36

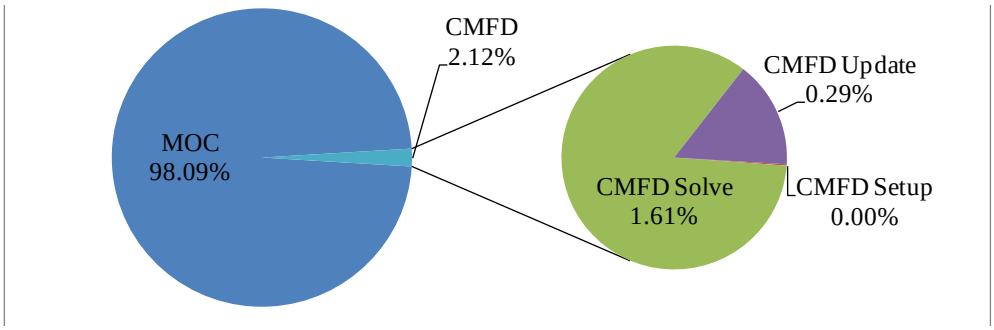


Figure 8.6 – Solution time breakdown with CMFD for Takeda benchmark (1 spatial domain)

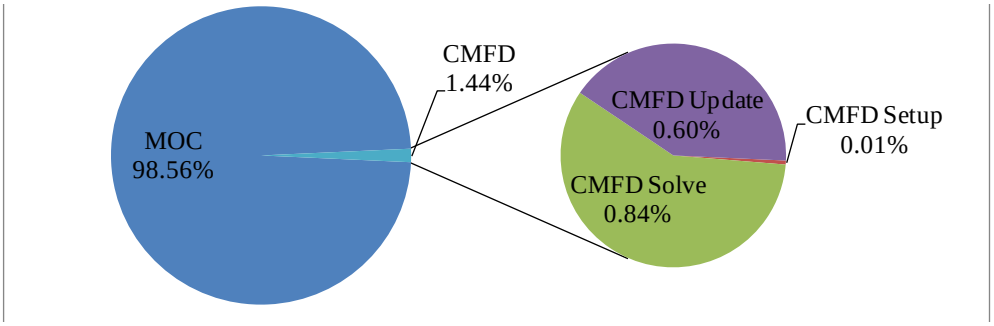


Figure 8.7 – Solution time breakdown with CMFD for Takeda benchmark (8 spatial domains)

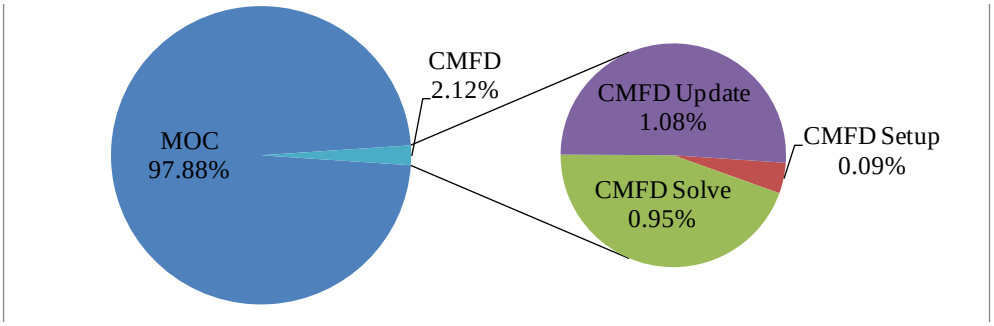


Figure 8.8 – Solution time breakdown with CMFD for Takeda benchmark (64 spatial domains)

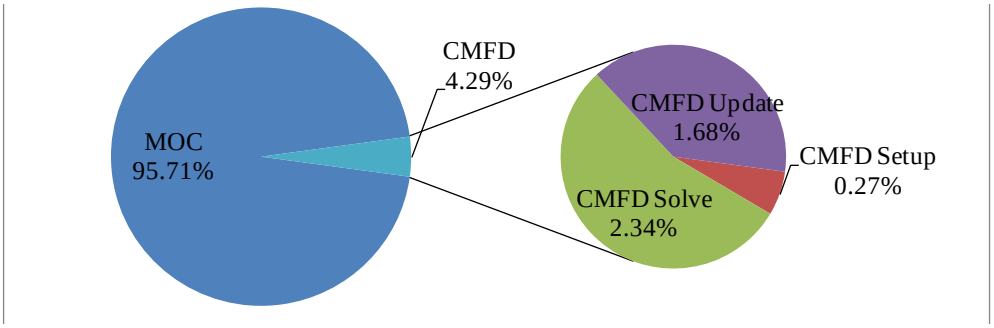


Figure 8.9 – Solution time breakdown with CMFD for Takeda benchmark (125 spatial domains)

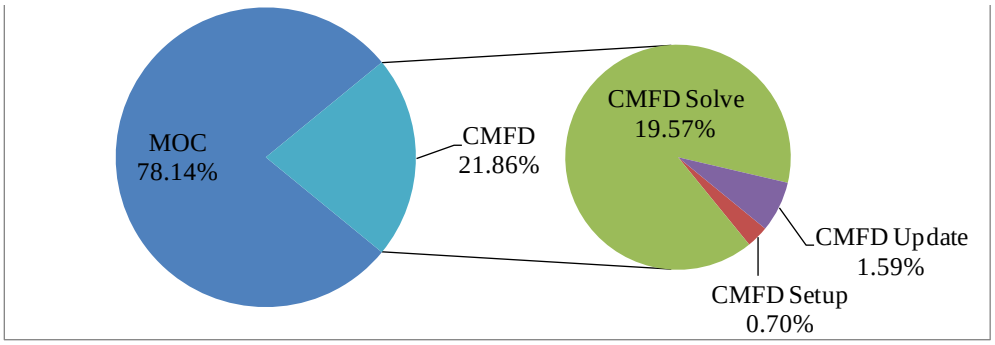


Figure 8.10 – Solution time breakdown with CMFD for Takeda benchmark (1000 spatial domains)

8.2 C5G7 Benchmark

The C5G7 benchmarks [69], [70] were an important addition to the set of publically available 3-D transport benchmarks because they were the first major benchmarks to include the detailed heterogeneity of reactor geometry for a modestly large domain. The original benchmark specified a 2-D and 3-D problem, and a second benchmark was created as an extension of the first which included multiple control rod configurations. The purpose of these benchmarks was to test the ability of modern deterministic transport codes to treat explicit reactor geometries without homogenization.

8.2.1 Model Description and Calculation Details

In the work here, the original 3-D benchmark was performed, as well as all three of the extended cases. The benchmark geometry shown in Figure 8.11 through Figure 8.13 is for the original 3-D benchmark description.

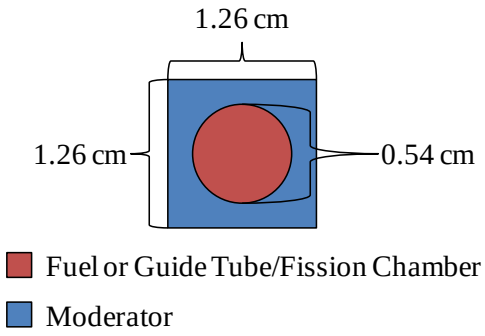


Figure 8.11 – C5G7 pin cell geometry

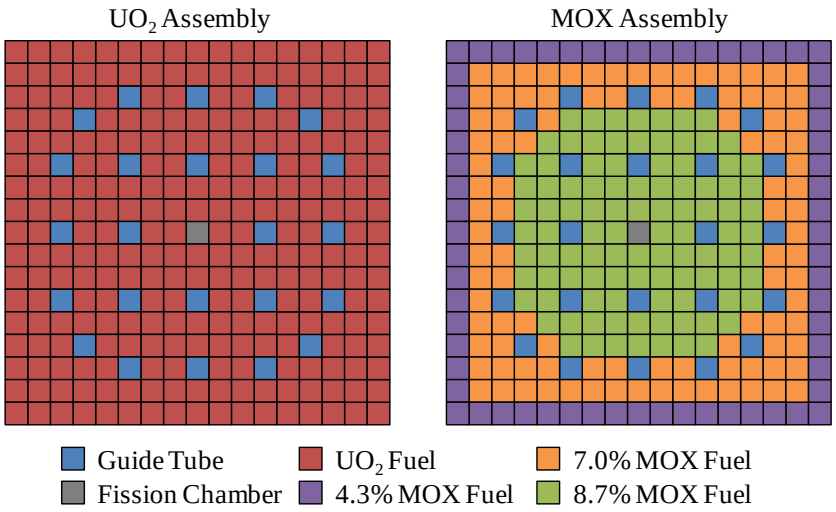


Figure 8.12 – C5G7 assembly descriptions

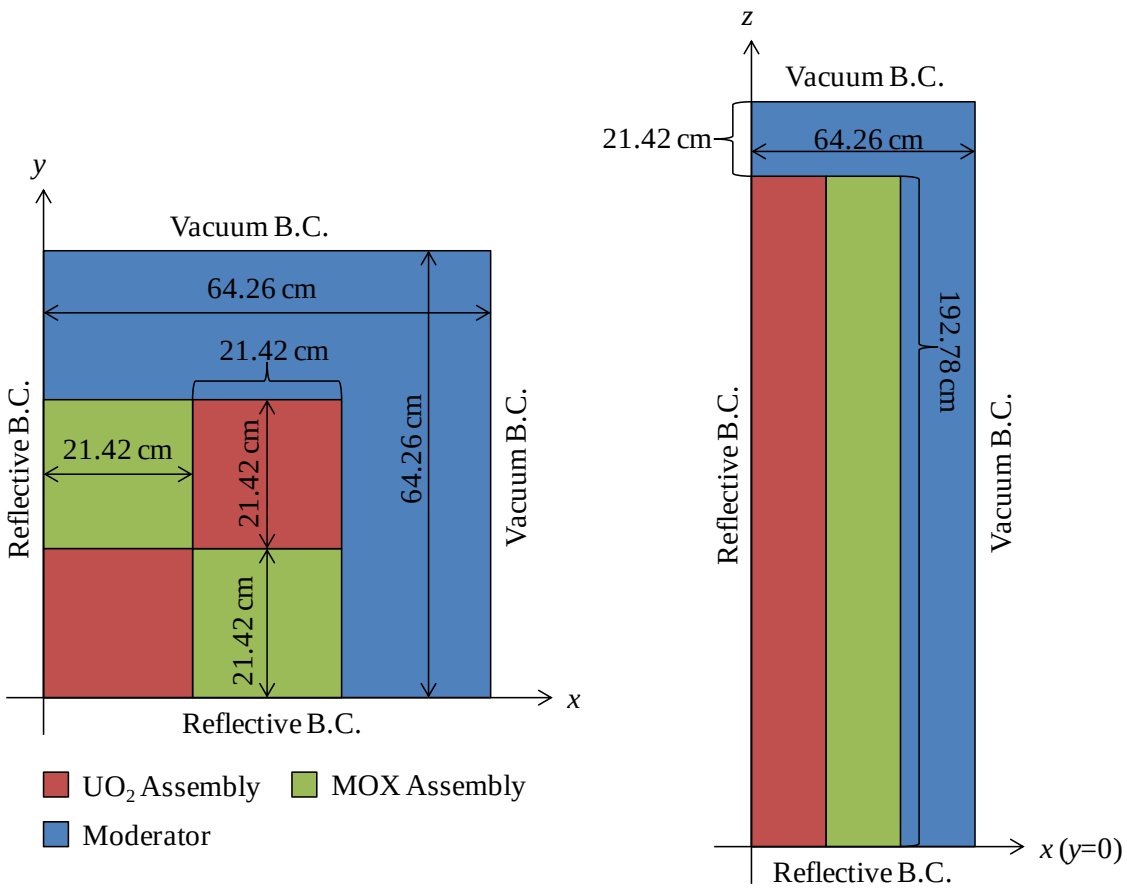


Figure 8.13 – C5G7 3-D core description

For the extended benchmark cases, which include control rods, the core geometry was reduced axially as shown in Figure 8.14. The geometry description for the pin cell (Figure 8.11) and assemblies (Figure 8.12) remain unchanged in the extended benchmark cases. The control rod positions for the three configurations: unrodded, rodded A, and rodded B are shown in Figure 8.15, Figure 8.16, and Figure 8.17, respectively. Figure 8.18 shows the layout of the rodded upper reflector, and in general when an assembly is rodded the control rod material replaces the guide tube material.

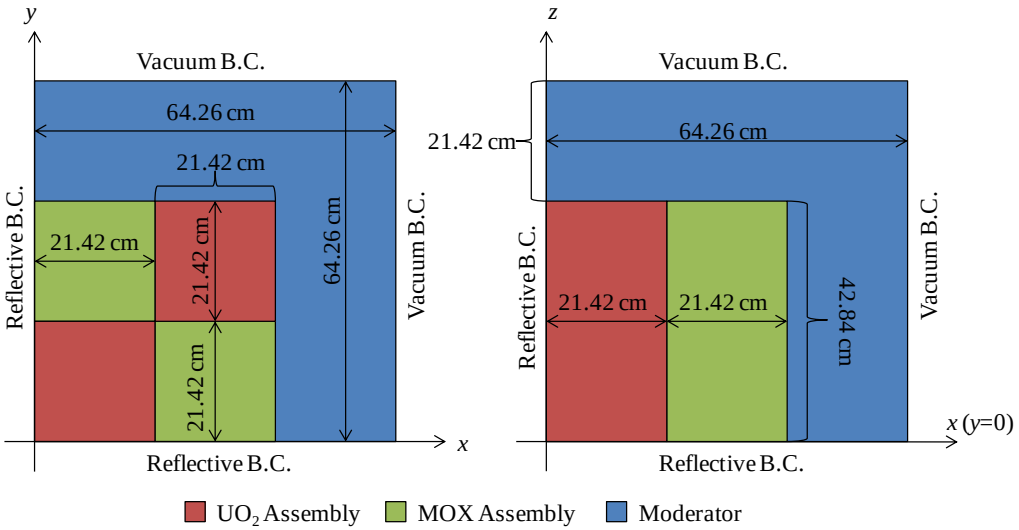


Figure 8.14 – C5G7 extended benchmark core description for rodded configurations

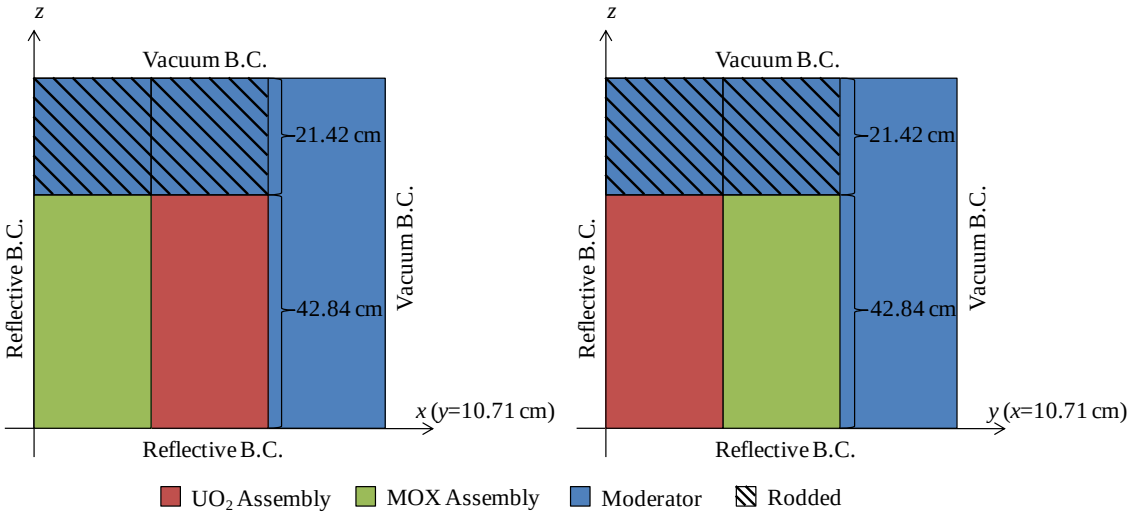


Figure 8.15 – C5G7 extended benchmark unrodded configuration

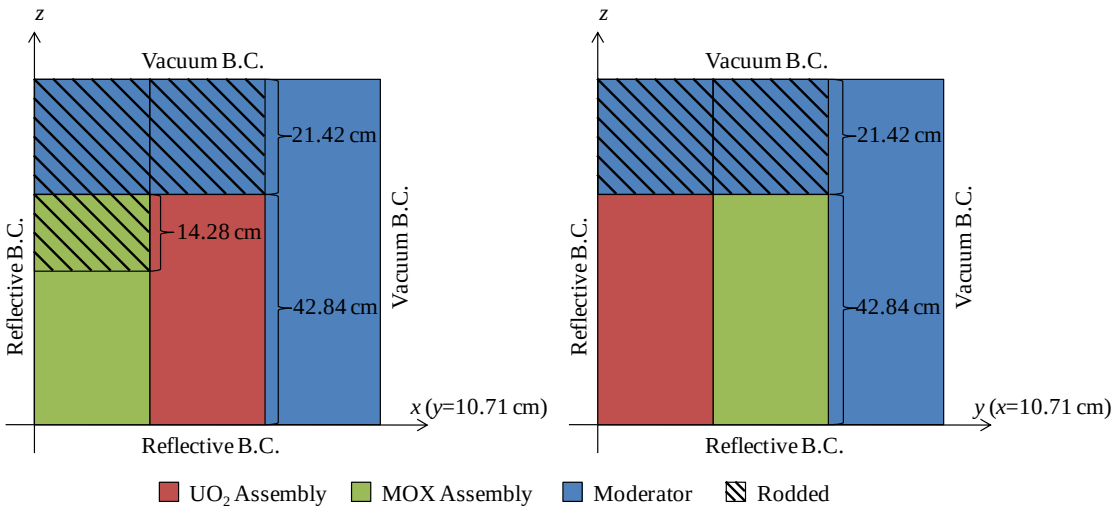


Figure 8.16 – C5G7 extended benchmark rodded A configuration

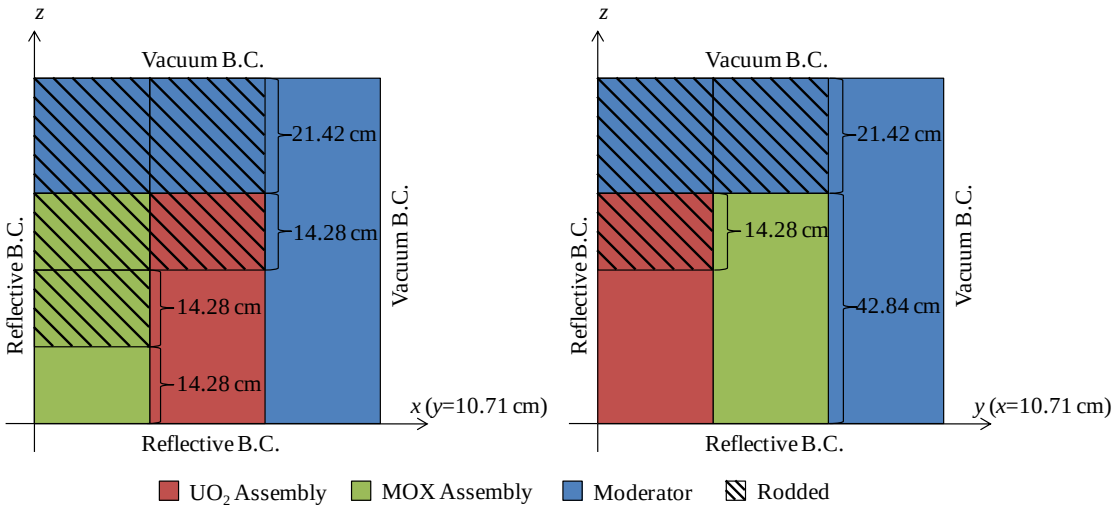


Figure 8.17 – C5G7 extended benchmark rodded B configuration

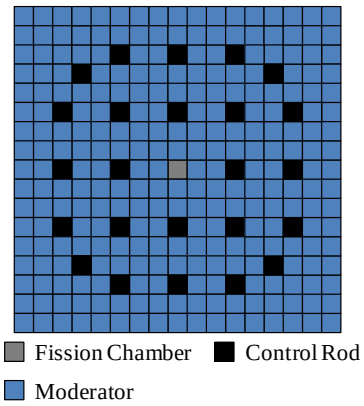


Figure 8.18 – Upper reflector assembly with control rod

Due to the increased problem size, a much smaller set of discretizations were examined compared to the Takeda problem. Furthermore, it was possible to choose the most appropriate discretizations based on previous experiences with the C5G7 benchmark using the 2-D/1-D solution scheme in previous work [69], [72]. For the spatial discretization the pin cells were discretized in the *x-y* plane using eight azimuthal sectors, five equal volume radial rings in the cylinder, and three equal volume rings in the moderator. The axial mesh in the pin cells was 0.252 cm and the reflector region was meshed using $(0.252\text{ cm})^3$ flat source regions. The dimensions of the modular ray tracing unit were 1.26 cm x 1.26 cm x 3.57 cm. The angular quadratures used were S_8 and S_{16}, which is the largest level symmetric quadrature order currently implemented. Ray spacings of 0.05 cm and 0.03 cm were also used. The spatial decomposition that was used for the original 3-D benchmark was 2160 domains, and the extended benchmarks used 648 spatial domains. Therefore, each process had approximately a quarter assembly sized domain, 3.57 cm tall. For all reported results 4 angular domains and 2 threads were used, and the convergence criteria used for Eq. (8.1) and Eq. (8.2) was 1.e-6 and 1.e-5, respectively.

8.2.2 Results and Discussion

In addition to the 3-D MOC results, a solution from the 2-D/1-D method was also calculated and compared to the benchmark reference. The results for 0.05 cm ray spacing and the S_8 quadrature are shown in Table 8.7, and the results for the 0.03 cm ray spacing and S_{16} quadrature are shown in Table 8.8. Results are also shown in Table 8.9 using 0.05 cm and a rectangular Chebyshev-Gauss (C-G) product quadrature, with 16 azimuthal directions and 4 polar directions per octant for the 3-D MOC and the 2-D MOC solution of the 2-D/1-D method. The computational cost for 2-D/1-D and 3-D MOC for the various benchmark cases are shown in Table 8.10.

Table 8.7 – Eigenvalue comparison for C5G7 3-D benchmark with S_8 quadrature and 0.05 cm ray spacing

Case	Reference k_{eff} (Δk_{eff} pcm)	3-D MOC (Diff. Δk_{eff} pcm)	2-D/1-D (Diff. Δk_{eff} pcm)
3-D Benchmark	1.183810 (±10)	1.18149 (-232)	1.18143 (-238)
Unrodded	1.14308 (±7)	1.14090 (-218)	1.13943 (-365)
Rodded A	1.12806 (±7)	1.12580 (-226)	1.12503 (-303)
Rodded B	1.07777 (±7)	1.07493 (-284)	1.07465 (-312)

Table 8.8 – Eigenvalue comparison for C5G7 3-D benchmarks with S_{16} quadrature and 0.03 cm ray spacing

Case	Reference k_{eff} (Δk_{eff} pcm)	3-D MOC (Diff. Δk_{eff} pcm)	2-D/1-D (Diff. Δk_{eff} pcm)
3-D Benchmark	1.183810 (±10)	1.18344 (-37)	1.18208 (-173)
Unrodded	1.14308 (±7)	1.14165 (-143)	1.14004 (-304)
Rodded A	1.12806 (±7)	1.12638 (-168)	1.12568 (-238)
Rodded B	1.07777 (±7)	1.07530 (-247)	1.07546 (-231)

Table 8.9 – Eigenvalue comparison for C5G7 3-D benchmarks with C_{16}-G_4 quadrature and 0.05 cm ray spacing

Case	Reference k_{eff} (Δk_{eff} pcm)	3-D MOC (Diff. Δk_{eff} pcm)	2-D/1-D (Diff. Δk_{eff} pcm)
3-D Benchmark	1.183810 (±10)	1.18250 (-131)	1.18383 (2)
Unrodded	1.14308 (±7)	1.14267 (-41)	1.14164 (-144)
Rodded A	1.12806 (±7)	1.12735 (-71)	1.12738 (-68)
Rodded B	1.07777 (±7)	1.07624 (-153)	1.07746 (-31)

Table 8.10 – Computational Cost for C5G7 3-D benchmarks with 2-D/1-D and 3-D MOC (CPU Hours)

Case	3-D MOC	2-D/1-D
3-D Benchmark	35884.80	1947.55
Unrodded	11805.12	108.55
Rodded A	10827.36	95.89
Rodded B	10085.76	108.13

Results for the S_8 quadrature are quite poor for both 3-D MOC and 2-D/1-D when compared to the benchmark reference. However, as the quadrature order is increased, the results improve, but it is clear that the S_{16} quadrature is still probably not a sufficient discretization for the rodded cases. The Chebyshev-Gauss quadrature gives surprisingly accurate results with the 2-D/1-D solver for the original 3-D case, but there is some inconsistent variation of the predicted eigenvalue for the rodded configurations. While the overall results may not be as accurate as the original benchmark participants, it is suggested that the primary reason for the differences observed in these results is from an

insufficient angular quadrature. It should be noted that this observation about the above results is consistent with the conclusions in [69], in which many of the participants used significantly higher order quadratures to obtain accurate results. The results from [69] are duplicated in Table 8.11 along with the angular quadrature order used by each participant, to illustrate this point. As for the computational cost, the CPU hours required for the 3-D MOC calculations are about a factor of 100 higher than the 2-D/1-D calculations for the rodded case, but only 20 times higher for the original 3-D benchmark.

Table 8.11 – C5G7 3-D benchmark participants' angular quadratures [69]

Code	Δk_{eff} (pcm)	Angular Quadrature
CRONOS2-SN	-658	S_4
TORT-GRS	-336	S_{16}
THREEDANT	11	S_8
DeCART	5	Chebyshev-Chebyshev Product Quadrature 8 azimuthal/8 polar per octant
CRX	155	Chebyshev-Chebyshev Product Quadrature 8 azimuthal/2 polar per octant
MCCG3D	-36	S_2 (1-D Gauss) for azimuthal 1 polar direction in 45°
PARTISN	-19	S_{26} square Chebyshev-Legendre
ATTILA	-33	S_{12} square Chebyshev-Double Legendre
TORT-ORNL	-147	S_{16}

In addition to the eigenvalue comparisons, summaries of the comparisons of the pin power for each case are shown in Table 8.12 through Table 8.15. These comparisons used the results of the S_{16} angular quadrature with 0.03 cm ray spacing. For the comparison of the 3-D benchmark the 3-D MOC eigenvalue is closer, but the errors in the pin power distribution are higher. For the extended cases the error in the solution of the 3-D MOC and 2-D/1-D are approximately the same in the unrodded case, however, once the rods enter the fuel region the 3-D MOC solution has less error compared to the reference than the 2-D/1-D. However, compared to the reference the 3-D MOC solution still shows some non-trivial error. The RMS error in the pin power for the 3-D MOC cases is about 0.5% in all cases. Based on this metric, these results place the 3-D MOC solution in about the middle of the spread of the original benchmark participants with 6 participants reporting higher RMS values, and 6 participants with less RMS error, and 2

participants with about the same error. Similar trends are observed for the metrics reported in Table 8.13 through Table 8.15. As noted earlier, the error in the 3-D MOC solution is attributed to the angular quadrature. Furthermore, it is noted that this quadrature does not provide the most accurate 2-D/1-D solution, therefore it is assumed that this is not the most accurate 3-D MOC solution. Thus, these results should be treated as preliminary and not representative of the most accurate results that are reasonably obtainable for this problem. That being said, these results when compared to the original benchmark participants results, are approximately in the middle, which is encouraging to the methods' overall accuracy.

Table 8.12 – C5G7 3-D benchmark pin power comparison

Metric	Axially Integrated Pin Powers		
Specific Pin Power Data	Reference	3-D MOC	2-D/1-D
Maximum Pin Power	2.500	2.515	2.504
Percent Error (associated 68% MC)	0.07	0.617	0.144
Distribution Percent Error Results			
Maximum Error (associated 68% MC)	0.190	1.457	2.045
AVG Error	0.139	0.469	0.376
RMS Error	0.145	0.563	0.504
MRE Error	0.118	0.461	0.296
Number of Accurate Fuel Pin Powers			
Number of Fuel Pins Within 68% MC	N/A	178	291
Number of Fuel Pins Within 95% MC	N/A	287	422
Number of Fuel Pins Within 99% MC	N/A	396	558
Number of Fuel Pins Within 99.9% MC	N/A	485	664
Total Number of Fuel Pins	1056	1056	1056
Average Pin Power In Each Assembly			
UO2-1 Power	1.867	1.876	1.869
MOX Power	0.802	0.798	0.801
UO2-2 Power	0.529	0.528	0.530
UO2-1 Power Percent Error	N/A	0.474	0.082
MOX Power Percent Error	N/A	-0.493	-0.175
UO2-2 Power Percent Error	N/A	-0.179	0.240

Table 8.13 – C5G7 pin power comparison for unrodded configuration

	Lower 1/3 of Fuel			Middle 1/3 of Fuel			Upper 1/3 of Fuel			Overall		
Specific Pin Power Data	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D
Maximum Pin Power	1.108	1.115	1.123	0.882	0.887	0.888	0.491	0.495	0.479	2.481	2.498	2.490
Percent Error (associated 68% MC)	0.090	0.593	1.336	0.100	0.594	0.706	0.130	0.916	-2.410	0.060	0.657	0.371
Distribution Percent Error Results												
Max. Error (associated 68% MC)	0.230	1.730	3.837	0.270	1.679	3.324	0.130	1.744	5.973	0.133	1.330	2.939
AVG Error	0.164	0.555	1.221	0.183	0.501	0.672	0.245	0.549	3.516	0.109	0.449	0.490
RMS Error	0.171	0.680	1.365	0.190	0.602	0.881	0.255	0.669	3.674	0.114	0.547	0.662
MRE Error	0.062	0.210	0.501	0.055	0.162	0.204	0.042	0.140	0.694	0.093	0.438	0.372
# of Accurate Fuel Pin Powers												
# of Fuel Pins Within 68% MC	371	73	8	371	103	88	371	154	3	371	88	67
# of Fuel Pins Within 95% MC	518	154	17	518	189	165	518	275	8	518	146	138
# of Fuel Pins Within 99% MC	540	223	29	540	246	215	540	335	12	540	178	179
# of Fuel Pins Within 99.9% MC	544	291	54	544	325	267	544	370	16	544	238	238
Avg. Assembly Power												
UO2-1 Power	219.04	219.62	221.49	174.24	174.88	175.24	97.93	98.95	95.04	491.21	493.45	491.77
MOX Power	94.53	93.89	95.48	75.25	74.81	75.54	42.92	43.02	41.10	212.70	211.72	212.12
UO2-2 Power	62.12	61.90	63.07	49.45	49.32	49.90	27.82	27.90	27.02	139.39	139.12	139.99
UO2-1 Power Percent Error	0.082	0.265	1.119	0.073	0.368	0.574	0.055	1.035	-2.951	0.123	0.455	0.114
MOX Power Percent Error	0.061	-0.678	0.998	0.054	-0.582	0.391	0.041	0.221	-4.242	0.092	-0.462	-0.274
UO2-2 Power Percent Error	0.043	-0.348	1.530	0.038	-0.266	0.907	0.029	0.286	-2.861	0.065	-0.192	0.433

Table 8.14 – C5G7 pin power comparison for rodded A configuration

	Lower 1/3 of Fuel			Middle 1/3 of Fuel			Upper 1/3 of Fuel			Overall		
Specific Pin Power Data	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D
aximum Pin Power	1.197	1.200	1.205	0.832	0.835	0.845	0.304	0.310	0.297	2.253	2.262	2.267
Percent Error (associated 68% MC)	0.080	0.218	0.674	0.100	0.430	1.614	0.200	2.021	-2.397	0.059	0.377	0.580
Distribution Percent Error Results												
Max. Error (associated 68% MC)	0.220	1.883	2.712	0.240	1.338	2.742	0.200	2.021	6.033	0.149	1.321	2.134
AVG Error	0.157	0.506	0.752	0.180	0.364	0.688	0.260	0.867	3.649	0.108	0.318	0.505
RMS Error	0.163	0.614	0.923	0.186	0.456	0.876	0.266	0.964	3.760	0.111	0.400	0.603
MRE Error	0.066	0.207	0.326	0.056	0.121	0.278	0.037	0.144	0.573	0.094	0.313	0.464
# of Accurate Fuel Pin Powers												
# of Fuel Pins Within 68% MC	371	74	40	371	156	111	371	56	1	371	123	45
# of Fuel Pins Within 95% MC	518	155	103	518	297	200	518	131	3	518	216	98
# of Fuel Pins Within 99% MC	540	216	135	540	361	243	540	187	6	540	266	142
# of Fuel Pins Within 99.9% MC	544	299	183	544	427	288	544	250	10	544	320	187
Avg. Assembly Power												
UO2-1 Power	237.41	237.71	238.96	167.51	167.94	169.47	56.26	56.98	54.26	461.18	462.62	462.68
MOX Power	104.48	103.79	105.05	78.01	77.69	78.21	39.23	39.51	37.62	221.71	220.99	220.88
UO2-2 Power	69.80	69.58	70.49	53.39	53.38	53.69	28.21	28.44	27.73	151.39	151.41	151.55
UO2-1 Power Percent Error	0.087	0.126	0.652	0.071	0.254	1.166	0.040	1.276	-3.555	0.119	0.312	0.326
MOX Power Percent Error	0.065	-0.658	0.550	0.056	0.408	0.261	0.040	1.276	-3.555	0.119	0.312	0.326
UO2-2 Power Percent Error	0.047	-0.313	0.984	0.040	-0.008	0.575	0.029	0.831	-2.957	0.068	0.008	0.105

Table 8.15 – C5G7 pin power comparison for rodded B configuration

	Lower 1/3 of Fuel			Middle 1/3 of Fuel			Upper 1/3 of Fuel			Overall		
Specific Pin Power Data	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D	Ref.	3-D MOC	2-D/1-D
Maximum Pin Power	1.200	1.193	1.217	0.554	0.556	0.556	0.217	0.220	0.209	1.835	1.837	1.840
Percent Error (associated 68% MC)	0.090	-0.559	1.427	0.150	0.394	0.428	0.240	1.396	-3.724	0.083	0.131	0.292
Distribution Percent Error Results												
Max. Error (associated 68% MC)	0.210	1.497	3.121	0.260	1.789	2.334	0.380	3.337	5.611	0.157	1.182	2.181
AVG Error	0.146	0.537	0.747	0.181	0.370	0.507	0.285	1.657	3.617	0.105	0.325	0.445
RMS Error	0.150	0.615	0.943	0.184	0.4484	0.681	0.290	1.714	0.3781	0.108	0.429	0.562
MRE Error	0.073	0.303	0.434	0.055	0.105	0.143	0.034	0.203	0.449	0.098	0.267	0.405
# of Accurate Fuel Pin Powers												
# of Fuel Pins Within 68% MC	371	56	74	371	150	126	371	0	2	371	109	62
# of Fuel Pins Within 95% MC	518	125	143	518	306	244	518	1	5	518	214	126
# of Fuel Pins Within 99% MC	540	173	182	540	377	315	540	13	7	540	286	169
# of Fuel Pins Within 99.9% MC	544	220	230	544	436	373	544	40	14	544	345	223
Avg. Assembly Power												
UO2-1 Power	247.75	246.50	250.43	106.56	106.80	106.18	41.12	41.73	39.69	395.43	395.02	396.30
MOX Power	125.78	124.95	126.21	81.41	81.46	81.70	29.42	29.88	28.16	236.62	236.29	236.07
UO2-2 Power	91.64	91.64	92.34	65.02	65.46	65.32	30.68	31.30	29.90	187.34	188.40	187.57
UO2-1 Power Percent Error	0.091	-0.505	1.084	0.056	0.226	0.359	0.035	1.470	-3.484	0.112	-0.103	0.220
MOX Power Percent Error	0.073	-0.666	0.341	0.058	0.063	0.357	0.034	1.559	-4.305	0.100	0.138	0.231
UO2-2 Power Percent Error	0.055	0.000	0.767	0.046	0.679	0.456	0.032	2.017	-2.523	0.078	0.566	0.120

Finally, some additional runs were executed on Titan to obtain more scaling information on the 3-D MOC kernel. These calculations did not run till convergence, instead only a few iterations were performed to obtain timing information. The original 3-D benchmark was fully decomposed in space to run on 156,060 processors and the unrodded extended benchmark case was fully decomposed in space to run on 46,818 processors. The subdomain sizes in each of these cases was a single pin cell so these data points represent the weak scaling in space of the 3-D MOC kernel. Figure 8.19 shows the parallel efficiency of the spatial decomposition weak scaling relative to a 2x2x2 pin cell case decomposed onto all 8 cores on a single NUMA node on Titan to provide a consistent reference. From the data in Figure 8.19 the spatial decomposition weak scaling is observed to be excellent with greater than 95% efficiency to $O(10^5)$ processors.

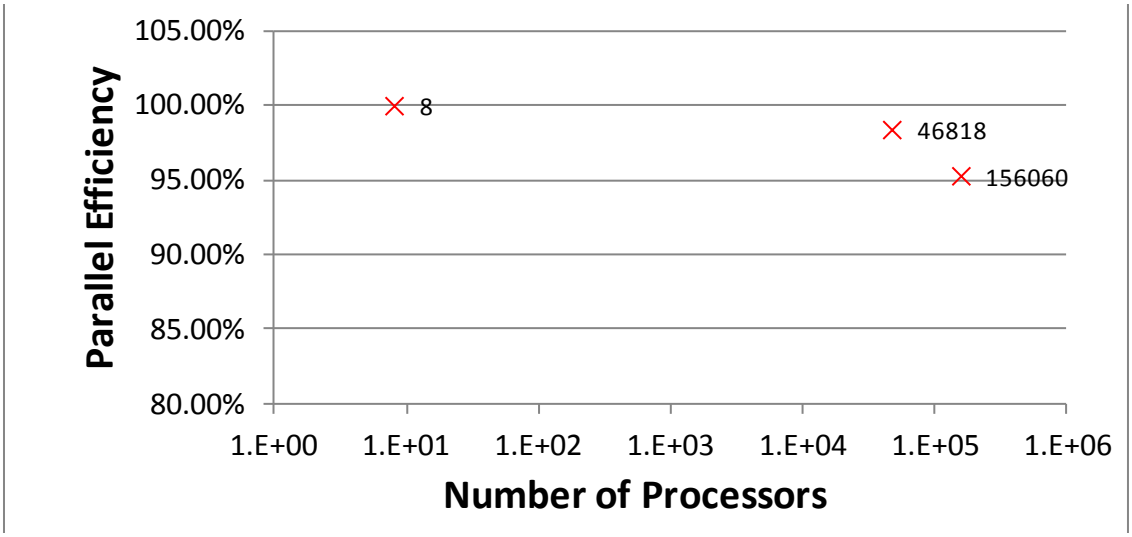


Figure 8.19 – Spatial decomposition weak scaling for C5G7

8.3 Realistic PWR Assembly

The problem studied in this section is adapted from the suite of CASL AMA Benchmarks [71]. In this suite there is a specification for a single 3-D PWR assembly, this is essentially the problem analyzed here, although some slight modification to the problem specifications is needed to accommodate a present limitation in the 3-D geometry modeling and modular ray tracing. This is described in more detail in the following section. The purpose of this benchmark is to begin to estimate the computational requirements of 3-D MOC and to evaluate the accuracy of 3-D MOC for a fairly realistic model. In addition to the increased geometric complexity of the model compared to the previous problems examined in this chapter, is the increased complexity of the cross sections and material descriptions in this model. The previous problems provided macroscopic cross sections in a few energy groups. However, in this problem the cross sections contain 60 energy groups, and the problem specifies only the material composition, so the macroscopic cross sections must be computed from this information and microscopic cross section data. A key step in this process is the resonance calculation which was performed here using the subgroup method [78]. Therefore, an additional feature in the analysis of this problem is the dependence of the resonance calculation on the dimensionality of the transport solution.

8.3.1 Model Description and Calculation Details

As mentioned previously, the model studied here deviates slightly from the specification in order to accommodate a limitation of the 3-D modular ray tracing implementation. This present limitation is that the modular ray tracing units must have a uniform height and there is currently no way to describe an axially heterogeneous material description in the ray tracing module. Therefore, the material boundaries of the model's geometry must align with the ray tracing modules. This requires that only the axial description of the original specification be modified. The modified axial description is shown in Figure 8.20. This limitation will be straightforward to eliminate in future work by allowing for axially heterogeneous material descriptions within the ray tracing module.

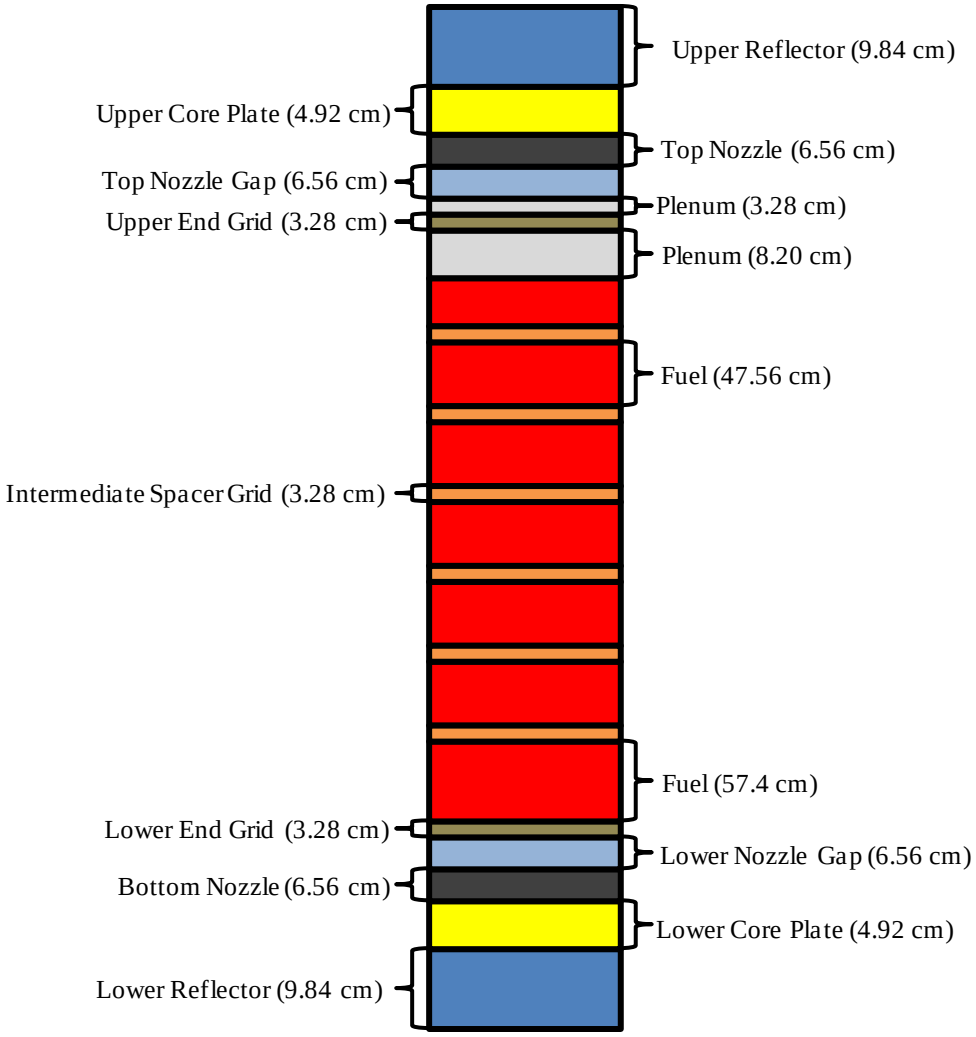


Figure 8.20 – Axial description of realistic PWR assembly (not to scale)

The radial description of the assembly is given in Figure 8.21. The assembly is a standard 17x17 PWR design. It has a uniform fuel enrichment of 3.1% U-235. The moderator contains 1300 ppm boron and the calculation is for hot zero power conditions. The model also includes a pellet-clad gap and the inter-assembly gap. The nozzles, core plate and reflector are each their own homogenous materials, and the grid spacers are homogenized into the coolant. The nozzle gap regions contain the guide tube structures. The rest of the detailed material compositions and geometry description are excluded for brevity, for the complete information it is suggested to consult the reference [71].

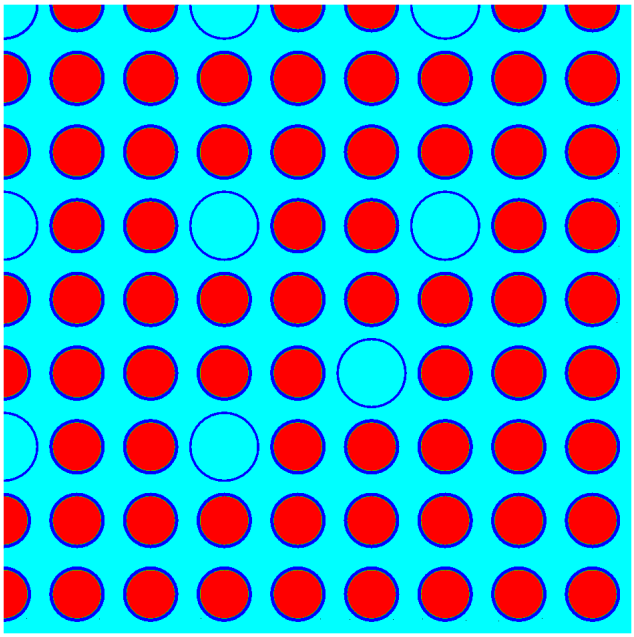


Figure 8.21 – Realistic PWR assembly radial geometry

Two models were developed, one for the 3-D MOC solver, and one for the 2-D/1-D solver. The cross sections are discretized into 60 neutron energy groups in both cases. The angular discretization used for the 2-D/1-D model was the Chebyshev-Gauss product quadrature with 16 azimuthal angles and 4 polar angles, and the S_{16} level symmetric quadrature was used for the 3-D MOC calculation. The ray spacing used was 0.05 cm. The models also made use of quarter symmetry and the ray tracing module dimensions were 10.75 cm x 10.75 cm x 1.64 cm. For the spatial discretization of the 3-D MOC model the fuel pins contained 224 flat source regions, the guide tubes had 128 flat source regions and the cells for the structural components and reflector had 64 flat source regions. The 2-D/1-D model used 56, 32, and 16 flat source regions for each pin cell type respectively. The difference is that the 3-D MOC model discretized each pin cell into four axial levels. The total number of flat source regions, segments and long rays is given in Table 8.16 for the 3-D MOC model. Additionally, the model was run with P_0 scattering rather than transport corrected P_0 scattering so that the 2-D/1-D model could converge.

The 3-D MOC model was run with a 261 spatial domains, 16 angular domains, and 4 threads for each MPI process for a total of 16,704 processors. The 2-D/1-D calculation was run with 46 spatial domains and 8 angular domains and 1 thread for each

MPI process for a total of 368 processors. The convergence criteria used was 1.e-5 for ε_{keff} and ε_{flux}.

Table 8.16 – Realistic PWR problem size parameters

Problem Size Parameter	3-D MOC Model Value	2-D/1-D Model Value
nseg	55,952,023,038	28,981,236
nlongray	2,238,077,088	407,008
nreg	3,697,984	157,496
nangoct	36	64

Finally, as a result of the slight modification to the axial description of the model, there is no longer a quality Monte Carlo reference solution for comparison. Consequently, there is only a comparison between the 3-D MOC result and a 2-D/1-D result.

8.3.2 Results and Discussion

Table 8.17 shows the computed k_{eff} for each case and the number of iterations to converge, as well as run time.

Table 8.17 – Comparison of realistic PWR problem

	3-D MOC	2-D/1-D	Difference
k_{eff}	1.17180	1.17323	143 pcm
No. of Iterations	7	18	11
Run Time	2103 s	630 s	1437 s

The primary difference in the k_{eff} is the different angular quadratures used for the two models. From the C5G7 results it was observed that the differences in these quadratures could easily account for 143 pcm. The ideal comparison would be to have the same angular quadrature for both, however, it was observed that the 2-D/1-D solver could not converge when using the S_{16} quadrature. This is likely due to a bug in the implementation of the 2-D/1-D solver. Additionally, the 3-D MOC could not successfully perform the ray tracing using the Chebyshev-Gauss quadrature, and that is likely caused by a different coding problem.

The number of iterations to converge is significantly higher for the 2-D/1-D solver and this is because the iteration scheme must be under-relaxed to remain stable when solving a problem like the realistic PWR assembly. Additionally, the 2-D/1-D

solver is using a DP_0 update for the angular flux boundary condition in the SDD-CMFD, while the 3-D MOC is using the P_1 update. It is remarkable though that the 3-D MOC solution is able to converge in just 7 iterations. This is the same number of iterations required to converge the Takeda problem which is drastically less complex. This is a testament to the effectiveness of the SDD-CMFD. Finally, despite the 3-D MOC using a factor of 45 more processors than the 2-D/1-D solution, it was still a factor of 3 slower in run time, which again highlights the computational burden of 3-D MOC.

8.4 Summary

In this chapter three numerical benchmark problems were evaluated with the parallel 3-D MOC kernel. The Takeda problem showed that the implementation of the 3-D MOC method is accurate and that the mesh converged solution is within the error of the benchmark reference. Additionally, the effectiveness of the SDD-CMFD was demonstrated and the parallel efficiency was demonstrated for this problem.

The C5G7 showed that the discretizations available for the 3-D MOC are not quite sufficient to achieve a mesh converged solution, but as the discretizations were refined the solution did approach the benchmark reference. This also provides confidence in the accuracy of the method. However, future work is needed to obtain solutions for more refined discretizations, particularly in angle, which will better show the accuracy and cost of the method compared to the other benchmark participants' results. The parallel efficiency for this problem was also observed to be greater than 90% with only spatial decomposition out to 156,060 processors.

Finally, a realistic PWR assembly was simulated and compared to 2-D/1-D results. The results of this problem contained some inconsistencies that cannot yet be resolved, thus making it difficult to draw any final conclusions. However, it was observed that the SDD-CMFD acceleration of the 3-D MOC was still very effective, even for such a complex problem. Furthermore, stability issues were encountered in the 2-D/1-D simulation, that are inherent to this method, and the 3-D MOC method had no stability issues. Also, the inconsistencies between the 3-D MOC and 2-D/1-D solutions is attributed to the differences in the models.

Chapter 9

SUMMARY, CONCLUSIONS, AND CONTINUING WORK

9.1 Summary of Work

This thesis began with the assertion that despite its computational intensity, the method of characteristics is a viable solution method for whole core, pin resolved, 3-D transport analysis of light water reactors. This is particularly true when compared to the other transport methods that have received significant attention for LWR application, such as the discrete ordinates method and the Monte Carlo method. However, each of these methods was shown to still have some outstanding issues for LWR applications that are currently being addressed.

In Chapter 2 the derivation of the 3-D MOC equations was presented, with particular attention to the important approximations that are commonly applied to 2-D MOC. It was also shown that after the transformation to the characteristic direction, the final form of the discrete MOC equations in 3-D are almost identical to the 2-D form. The primary difference in 2-D and 3-D MOC is the discretization of the problem. One of the original aspects of the research here is the extension of the modular ray tracing concept from 2-D MOC to 3-D MOC in a way that was slightly different from previous work and that reduces the number of angles that must be stored for 3-D MOC.

In Chapter 3, the specific details of the implementation of the MOC equations were presented. This chapter leveraged the considerable progress made in previous work on 2-D MOC methods over the last several years, which helped ensure that the serial performance of the 3-D MOC kernel is nearly as good as what is observed for 2-D MOC kernels. The other purpose of presenting this detail is to remove any ambiguity of how

the equations are mapped to the programming model, which was an essential component needed for the work done in Chapter 4, Chapter 5, and Chapter 6.

Once a clear description of the 3-D MOC kernel was established, the methods were presented for parallelizing the kernel in Chapter 4. The parallelization used both shared and distributed memory models and included three types of decomposition. The parallelization of the space and angle phase space was performed using a distributed memory model. The communication pattern for the spatial decomposition involved using non-blocking point-to-point communication between neighboring domains, which is required to exchange the boundary condition between the domains. The communication pattern for the angular decomposition involved an all reduce operation to compute the scalar flux from partial sums accumulated on each angular domain. The remaining decomposition for the characteristic rays used the shared memory model for very fine grained parallelism.

For each decomposition the methods by which the domains were partitioned was also described in Chapter 4. For the spatial domain, the concept of modular ray tracing was again leveraged, since it superimposes a structured Cartesian grid over the problem domain, which simplifies the partitioning algorithm. The Z-order space filling curve was used to define a tree data structure through which this grid is indexed and partitioned. The use of a space filling curve helps to preserve several properties that are advantageous for achieving good parallel performance. The angular domain was partitioned using a greedy algorithm, since the work for each angle may vary considerably, and without a good load balance from the partitioning it was not possible to achieve acceptable parallel performance. The ray decomposition that was implemented with OpenMP uses a built in library routine for dynamic scheduling, and the MPI standard was used to implement the distributed memory parallelism.

The next phase in the research, was the development of a performance model capable of predicting the execution time and other performance metrics of the MOC kernel as a function of the problem size input and computer architecture hardware properties. The model was validated and used to predict and to analyze the performance of the MOC kernel for a wide range of problem inputs on a wide variety of architectures

without having to actually execute the code. The development and application of this model was another original aspect of the research performed as a part of this thesis.

Chapter 5 presented the experimental validation of the performance model for serial execution, and the experimental procedure and measurement system for collecting the data to validate the model was described in detail. The methods used to measure the hardware properties required to evaluate the performance model were also presented. The experimental measured values and the computed values of the model were compared for several metrics. The model was shown to predict the number of FLOPs exactly, and the number of loads to within about 12% of the measured values. When the cache misses were known, the execution time predicted by the model was shown to agree with measurement to within 8%. This was bounded by the assumed cache misses, where the lower bound is zero and the upper bound assumes a cache miss at every level of cache for each load. The serial performance of the kernel was base lined between 440 and 490 MFLOPS, which was approximately 4.5% of the machine's theoretical peak performance and 50% of the kernel's theoretical upper bound for performance.

In Chapter 6 the performance model was applied to parallel performance and first validated against experiments for the prediction of the parallel efficiency and speedup. This was done separately for each type of parallel decomposition. Again, the experimental procedure and methods were described for measuring the coefficients of the hardware properties required by the model. The performance model was shown to predict the parallel efficiency to within 7% (absolute difference) of the measured parallel efficiency for all decompositions. It was noted that the model assumes a perfect scaling in the hardware, whereas this may not always be the case. Furthermore, the machine used to collect the performance data has only one floating point arithmetic unit for every two cores, and this raised questions about what is the most meaningful or consistent choice to use as a reference calculation when measuring parallel efficiency.

Once the validation of the model was established for parallel execution, the remainder of the Chapter 6 focused on the analysis of the sensitivity of this model to the network hardware characteristics and OpenMP run time library routines. It was found that the overhead for space or angle decomposition is relatively insensitive to the network hardware properties, and that networks on today's high performance compute clusters are

probably already sufficiently fast to achieve the asymptotic lower bound of the overhead. The algorithm used for the `MPI_Allreduce` operation was shown to have a nontrivial impact on the angular decomposition overhead, and algorithms that are more optimal for large message sizes were shown to have a lower overhead. When analyzing the ray decomposition's sensitivity to the OpenMP run time library, it was found that the bottle neck limiting parallel efficiency is the barriers for synchronization. The model was then evaluated for various parallel decompositions in an attempt to understand the decomposition strategies for achieving optimal parallel efficiencies. It was determined that it was most important to optimize the spatial decomposition first, and then the optimal angular and ray decomposition depended on how finely in space the problem was decomposed. For typical discretizations of a pin cell, it was noted that a spatial subdomain consisting of 2x2x2 or 4x4x4 pin cells can be efficiently parallelized with an additional 16 processors. The strong scaling predicted by the performance model for a quarter core sized problem was also evaluated and determined to scale to nearly 30 million processors, with a parallel efficiency of at least 90%.

The next phase of the research was to focus on the optimization of the solution algorithm. Based on the considerable previous research in methods for accelerating the transport equation, the focus of the work here was on the well-established coarse mesh finite difference (CMFD) method. Chapter 7 began with a description of the CMFD method as a non-linear diffusion synthetic acceleration, which has been shown to perform well for reactor problems. Other researchers have also recently extended CMFD to problems that are decomposed in space. This method was implemented and shown to reproduce the convergence properties of model problems in 1-D, and that the convergence properties for 3-D transport with heterogeneous domains behaved similarly.

Finally, in Chapter 8 the 3-D MOC kernel was used to perform several numerical 3-D transport benchmarks commonly used in the reactor physics community. Three problems were investigated, and for the first benchmark, the Takeda benchmark, the 3-D MOC method was shown to provide excellent accuracy and to scale well on several thousand processers. The effectiveness of the CMFD was also shown to be very good, reducing the number of iterations by an order of magnitude for most decompositions. It

was also shown that the number of iterations to converge with CMFD was relatively insensitive to the number of spatial domains.

The results of the next benchmark, the C5G7 heterogeneous MOX core, were not as good as the Takeda benchmark, but still provided reasonable accuracy. The principal issues uncovered were the need for a higher order angular quadrature to improve the accuracy and that the product quadratures tended to perform a little better than the level symmetric quadrature, which was consistent with the observations from the original benchmark report. For this problem the accuracy and computational cost of the 3-D MOC method was also compared to the 2-D/1-D solution method. It was noted that the computational cost of the 3-D MOC was considerably higher (i.e. 20x to 200x that of the 2-D/1-D method), and the accuracy was shown to be only marginally better for consistent discretizations. Finally, with the C5G7 problem the parallel efficiency of the 3-D MOC kernel was demonstrated to scale to $O(10^5)$ processors with >95% efficiency.

For the last benchmark problem of a realistic PWR assembly, the model had to deviate slightly from the original benchmark specification to accommodate a modeling deficiency in the code. This deviation only required changing the relative heights of the different material regions. The 3-D MOC results were again compared to 2-D/1-D results and the differences in eigenvalue were minor. The 3-D MOC case converged very efficiently and the run time was approximately 35 minutes on 16704 processors while the 2-D/1-D case ran in 10 minutes on 368 processors.

9.2 Suggested Future Research

The work performed in this research has provided some original insights that have contributed to the current understanding of the 3-D MOC. However, the overall understanding and experience with 3-D MOC is still far less than that of other 3-D transport methods, or even the 2-D MOC, for which there is an extensive research base. Based on the research here and the observations of other researchers, there are several areas of future research that are important to further establish the viability of the parallel 3-D MOC for practical reactor analysis.

9.2.1 Modular Rays and Angular Quadratures

The first area of recommended future research is related to the construction of the modular rays. In Chapter 2 it was noted that the method used in this work could be improved, as it was not necessarily an improvement over previous methods. It is suggested that further research be performed to investigate more optimal methods of determining the modular ray directions and spacing. Closely related to this is the choice of a starting quadrature for the discrete ordinates. The results from Chapter 8 suggest that product quadratures may be better for reactor problems, and therefore one area of future work should be to investigate in more detail optimal quadratures for LWR applications. Additionally, in the current work the renormalization of the weights of the discrete ordinates has only been demonstrated for the level symmetric quadrature. More generalized techniques for recomputing the weights that can be applied to product quadratures should also be investigated in future work. Another alternative is to attempt to develop new angular quadratures that are already modular.

9.2.2 Spatially Higher Order Sources

Another suggested area of future research is to investigate higher order sources. It has been shown in previous work [73] that using a linear source in 2-D can substantially reduce requirements for the spatial discretization. For 3-D problems, it has been shown to provide speedup factor of about a factor of 8 [74]. It is expected that similar results can be obtained for the implementation performed here. Additionally, it may be worthwhile to investigate quadratic sources or sources that are only linear in the axial direction as a more optimal way of representing the source.

9.2.3 Acceleration Techniques

Considerable work has been done within the transport community to investigate acceleration techniques. In particular, there is the class of methods for boundary projection acceleration [59], and also a similar class of methods, known as DP_N acceleration [75], that may better accelerate the interface angular fluxes on spatial subdomains. Acceleration techniques that are higher order in angle, such as an angularly

dependent form of CMFD or coarse mesh rebalance (CMR) [76], could also be better than traditional CMFD for accelerating the subdomain interface angular fluxes.

9.2.4 Optimizing Performance and Parallelism for Energy Groups

Further research is suggested on the overall performance and parallelism of the 3D MOC kernel. Recently, it was shown that moving the loop over groups to be the inner-most loop, rather than the outermost loop can provide better performance [77]. This is consistent with the observations in Chapter 5, which showed that nearly half the kernel time was spent building the long ray information, and considerable savings could be achieved by inverting the looping structure for the energy groups instead of rebuilding each long ray for each group. This could lead to a potential speedup by a factor of G, where G, is the number of neutron energy groups. However, this would change the convergence properties of the scattering source, which may require more iterations to converge. If this iteration scheme is shown to still converge reasonably well with an acceleration technique, then it could be a significant step forward in improving performance. The result of this research would also be important for guiding the research into parallelization over the energy domain.

9.2.5 Mapping to GPU architectures

The final area to consider for future work is modifying the kernel's on-node problem to execute on a GPU instead of a multi-core processor. Many high performance compute clusters are currently moving towards this type of heterogeneous architecture. This research would be important if the 3-D MOC algorithm is going to be able to take advantage of the near-term future architectures.

9.3 Final Remarks

The research in this thesis has extended the state of the current knowledge of 3-D MOC and its application to reactor problems. The goal of developing a highly scalable parallel algorithm for 3-D MOC was achieved, and many of the ideas developed and explored for the parallelism and performance modeling can be applied in a relatively straightforward manner to other transport methods. It is still not clear which 3-D transport method is best

for whole core pin resolved LWR analysis, but as research in this area continues, there is now a well defined and quantifiable method to establish the cost, performance, and accuracy of the 3-D MOC method, which can help establish the basis for a consistent comparison and clarify the most important issues.

BIBLIOGRAPHY

[1] G. C. BILODEAU, W.R. CALDWELL, J.P. DORSEY, J.G. FAIREY, and R.S. VARGA, "PDQ-An IBM-704 Code to Solve the Two-dimensional Few-group Neutron Diffusion Equations," Tech. Rep. WAPD-TM-70, Bettis Plant, Westinghouse Electric Corporation (1957).

[2] K.S. SMITH and J.D. RHODES, "CASMO-4 Characteristics Methods for Two-Dimensional PWR and BWR Core Calculations," *Transactions of the American Nuclear Societcy*, **83**, 294 (2000).

[3] R. SANCHEZ, I. ZMIJAREVIC, M. COSTE-DELCLAUX, E. MASIELLO, S. SANTANDREA, E. MARTINOLLI, L. VILLATE, N. SCHWARTZ, and N. GULER, "APOLLO2 Year 2010," *Journal of Nuclear Engineering and Technology*, **42**, 5, 474-499 (2010).

[4] H.G. JOO, J.Y. CHO, K.S. KIM, C.C. LEE, and S.Q. ZEE, "Methods and Performance of a Three-Dimensional Whole-Core Transport Code DeCART," *Proceedings of PHYSOR 2004 - ThePhysics of Fuel CYcles and Advanced Nuclear Systems*, Chicago, IL, USA, April 25-29 (2004), [CD-ROM].

[5] S.G. HONG and N.Z. CHO, "CRX: A Code for Rectangular and Hexagonal Lattices Based on the Method of Characteristics," *Annals of Nuclear Energy*, **25**, 547-565 (1998).

[6] T.M. EVANS, A.S. STAFFORD, R.N. SLAYBAUGH, and K.T. CLARNO, "Denovo: New Three-Dimensional Parallel Discrete Ordinates Code in SCALE," *Journal of Nuclear Technology*, **171**, 2, 171-200 (2010).

[7] W.S. YANG, M.A. SMITH, C.H. LEE, A.B. WOLLABER, D. KAUSHIK, and A.S. MOHAMED, "Neutronics Modeling and Simulations of SHARP for Fast Reactor Analysis," *Journal of Nuclear Engineering and Technology*, **42**, 12, 520-545 (2010).

[8] K. S. SMITH, "Monte Carlo for Practical LWR Analysis: What's Needed to Get to the Goal," *Transactions of the American Nuclear Society*, **104**, 313 (2011).

[9] D.J. KELLY, T.M. SUTTON, and S.C. WILSON, "MC21 Analysis of the Nuclear Energy Agency Monte Carlo Performance Benchmark Problem," *Proceedings of PHYSOR 2012 - Advances in Reactor Physics - Linking Research, Industry, and Education*, Knoxville, TN, USA, April 15-20 (2012), [CD-ROM].

[10] "Consortium for Advanced Simulation of Light Water Reactors (CASL)," http://www.casl.gov/.

[11] K.S. SMITH and B. FORGET, "Challenges in the Development of High-Fidelity LWR Core Neutronics Tools," *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2013)*, Sun Valley, ID, USA, May 5-9 (2013), [CD-ROM].

[12] M.J. LEE, H.G. JOO, D.J. LEE, K.S. SMITH, "Monte Carlo Reactor Calculation with Substantially Reduced Number of Cycles," *Proceedings of PHYSOR 2012 - Advances in Reactor Physics - Linking Research, Industry, and Education*, Knoxville, TN, USA, April 15-20 (2012), [CD-ROM].

[13] G. YESILYURT, W.R. MARTIN, and F.B. BROWN, "On-the-Fly Doppler Broadening for Monte Carlo Codes," *Journal of Nuclear Science and Engineering*, **171**, 239-257 (2012).

[14] J.E. HOOGENBOOM, W.R. MARTIN, and B. PETROVIC, "The Monte Carlo Benchmark Test - Aims, Specifications, and First Results," *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2011)*, Rio de Janeiro, Brazil, May 8-12 (2011), [CD-ROM].

[15] E.E. LEWIS and W.F. MILLER, JR., *Computational Methods of Neutron Transport*, Wiley-Interscience (1984).

[16] K.R. KOCH, R.S. BAKER, and R.E. ALCOUFFE, "Solution of the First-Order Form of Three-Dimensional Discrete Ordinates Equations on a Massively Parallel Machine," *Transactions of the American Nuclear Society*, **65**, 198-199 (1992).

[17] Oak Ridge Leadership Computing Facility, "Oak Ridge Leadership Computing Facility," (2012), http://www.olcf.ornl.gov/computing-resources/jaguar/.

[18] S. BALAY, J. BROWN, K. BUSCHELMAN, W.D. GROPP, D. KAUSHIK, M.G. KNIPLEY, L.C. McINNES, B.F. SMITH, and H. ZHANG, "PETSc Web page," (2013), http://www.mcs.anl.gov/petsc/.

[19] Argonne National Laboratory, "Intrepid - Argonne Leadership Computing Facility," (2013), http://www.alcf.anl.gov/intrepid.

[20] Jülich Supercomputing Center, "Forschungszentrum Jülich - JUGENE", (2012), http://www.fz-juelich.de/jsc/jugene.

[21] D. KAUSHIK, M.A. SMITH, A.B. WOLLABER, B.F. SMITH, A. SIEGEL, and W.S. YANG, "Enabling High-Fidelity Neutron Transport Simulations on Petascale Architectures," *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, Portland, OR, USA, November 14-20 (2009).

[22] S.E. KELLER and C.R.E. DE OLIVEIRA, "Two-dimensional C5G7 MOX Fuel Assembly Benchmark Calculations using the FEM-PN code EVENT," *Progress in Nuclear Energy*, **45**, 2-4, 255-263, (2004).

[23] R.J.J. STAMM'LER and M.J. ABBATE, *Methods of Steady-State Reactor Physics in Nuclear Design*, Academic Press (1983).

[24] E.A. VILLARINO, R.J.J. STAMM'LER, A.A. FERRI, and J.J. CASAL, "HELIOS: Angularly Dependent Collision Probabilities," *Journal of Nuclear Science and Engineering*, **112**, 16-31 (1992).

[25] D.B. JONES, K.E. WATKINS, and M.L. WILLIAMS, "CPM3 Computer Code Manual, vol 1, Theory and Numerics Manual," Tech. Rep. EPR-CPM-001-M-001, Revision A, Electric Power Research Institute (2000).

[26] G. MARLEAU, R. ROY, and A. HÉBERT, "DRAGON: A Collision Probability Transport Code for Cell and Supercell Calculations," Tech. Rep. IGE-157, Institut de génie nucléaire, Ecole Polytechnique de Montréal (1994).

[27] Global Nuclear Fuels, "LANCR02 Lattice Physics Model Description," Tech. Rep. NEDO-33376 Rev. 1, Nuclear Regulatory Commission (June 2009).

[28] W.M. STACEY, *Nuclear Reactor Physics*, 2nd Edition, Wiley-VCH (2007).

[29] J. ASKEW, "A Characteristics Formulation of the Neutron Transport Equation in Complicated Geometries," Tech. Rep. AEEW-R-1108, United Kingdom Atomic Energy Authority (1972).

[30] M.J. HALSALL, "CACTUS, A Characteristics Solution to the Neutron Transport Equations in Complicated Geometries," Tech. Rep. AEEW-R-1291, United Kingdom Atomic Energy Authority (1980).

[31] G.J. WU and R. ROY, "A new characteristics algorithm for 3D transport calculations," *Annals of Nuclear Energy*, **30**, 1-16 (2003).

[32] J. B. TAYLOR, D. KNOTT, and A.J. BARATTA, "A Method of Characteristics Solution to the OECD/NEA 3D Neutron Transport Benchmark Problem," *Proceedings of the Joint International Topical Meeting on Mathematics and Computation and Supercomputing in Nuclear Applications (M&C+SNA 2007)*, Monterey, CA, USA, April 15-19 (2007), [CD-ROM].

[33] Z. LIU, H. WU, L. CAO, Q. CHEN, and Y. LI, "A new three-dimensional method of characteristics for the neutron transport calculation," *Annals of Nuclear Energy*, **31**, 447-454 (2011).

[34] M. DAHMANI, G.J. WU, R. ROY, and J. KOCLAS, "Development and Parallelization of the Three-Dimensional Characteristics Solver MCI of DRAGON," *Proceedings of PHYSOR 2002*, Seoul, Korea, October 7-10 (2002), [CD-ROM].

[35] G.C. POMRANING, "Asymptotic and Variational Derivations of the Simplified P_N Equations," *Annals of Nuclear Energy*, **20**, 623-637 (1993).

[36] D. LEE, T. KOZLOWSKI, T. DOWNAR, C. LEE, and H.C. LEE, "Application of SP_3 Approximation to MOX Transient Analysis in PARCS," *Transactions of the American Nuclear Society*, **91**, 252-256 (2004).

[37] M. TATSUMI and A. YAMAMOTO, "Advanced PWR Core Calculation Based on Multi-group Nodal-Transport Method in Three-dimensional Pin-by-Pin Geometry," *Journal of Nuclear Science and Technology*, **40**, 376-387 (2003).

[38] J.Y. CHO, H.G. JOO, K.S. KIM, and S.Q. ZEE, "Three-Dimensional Heterogeneous Whole Core Transport Calculation Employing Planar MOC Solutions," *Transactions of the American Nuclear Society*, **87**, 234-236 (2002).

[39] N.Z. CHO, G.S. LEE, and C.J. PARK, "A Fusion Technique of 2-D/1-D Methods for Three-Dimensional Whole-Core Transport Calculations," *Proceedings of the Korean Nuclear Society*, Kwangju, Korea (May, 2002).

[40] B.W. KELLEY and E.W. LARSEN, "2D/1D Approximations to the 3D Neutron Transport Equation. I: Theory," *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2013)*, Sun Valley, ID, USA, May 5-9 (2013), [CD-ROM].

[41] P.K. ROMANO and B. FORGET, "Parallel Fission Bank Algorithms in Monte Carlo Criticality Calculations," *Journal of Nuclear Science and Technology*, **170**, 125-135 (2012).

[42] B. KOCHUNAS, T.J. DOWNAR, S. MOHAMED, and J.W. THOMAS, "Improved Parallelization of the Modular Ray Tracing in the Method of Characteristics Code DeCART," *Proceedings of the Joint International Topical Meeting on Mathematics and Computation and Supercomputing in Nuclear Applications (M&C+SNA 2007)*, Monterey, CA, USA, April 15-19 (2007), [CD-ROM].

[43] A. YAMAMOTO, Y. KITAMURA, Y. YAMANE, "Computational efficiencies of approximated exponential functions for transport calculations of the characteristics method," *Annals of Nuclear Energy*, **31**, 1027-1037, (2004).

[44] OpenMP Architecture Review Board "OpenMP Application Program Interface Version 3.1," http://openmp.org/wp/openmp-specifications/, July (2011).

[45] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard Version 2.2," http://www.mpi-forum.org/docs/mpi-2.2/, September (2009).

[46] A. SIEGEL, "High Performance Computing and Multiphysics Methods," (Presentation at MeV School, Argonne National Laboratory, Argonne, IL, July 26, 2011).

[47] S. KOSAKA and E. SAJI, "Transport Theory Calculation for a Heterogeneous Multi-Assembly Problem by Characteristics Method with Direct Neutron Path Linking Technique," *Journal of Nuclear Science and Technology*, **37**, 12, 1015-1023 (2000).

[48] M. BADER, *Space-Filling Curves: An Introduction with Applications in Scientific Computing*, Springer (2013).

[49] R.W. VUDUC, *Automatic Performance Tuning of Sparse Matrix Kernels*, Ph.D. thesis, University of California, Berkeley (2003).

[50] R. THAKUR, R. RABENSEIFNER, and W.D. GROPP, "Optimization of Collective Communication Operations in MPICH," *International Journal of High Performance Computing Applications*, **19**, 1, 49-66 (2005).

[51] R. RABENSEIFNER, "Optimization of Collective Reduction Operations," *Computational Science - ICCS 2004*, Springer (2004).

[52] S. BROWNE, J. DONGARRA, N. GARNER, G. HO, and P. MUCCI, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *International Journal of High Performance Computing Applications*, **14**, 3, 189-204 (2004).

[53] R.H. SAAVEEDRA-BARRERA, *CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking*, Ph.D. thesis, University of California, Berkeley (1992).

[54] Oak Ridge Leadership Computing Facility, "Introducting Titan - The World's #1 Open Science Superomputer," (2013), http://www.olcf.ornl.gov/titan/.

[55] D.K. PANDA, "OSU Micro-Benchmarks," (2013), http://mvapich.cse.ohio-state.edu/benchmarks/.

[56] J.M. BULL and F. REID, "OpenMP Microbenchmarks V2.0," (2013), http://www2.epcc.ed.ac.uk/computing/research_activities/openmpbench/openmp_index.html.

[57] J.M. BULL, "Measuring Synchronisation and Scheduling Overheads in OpenMP," *European Workshop on OpenMP (EWOMP '99)*, Lund, Sweden, October (1999).

[58] M.L. ADAMS and E.W. LARSEN, "Fast iterative methods for discrete-ordinates particle transport calculations," *Progress in Nuclear Energy*, **40**, 1, 3-159, (2002).

[59] M.L. ADAMS and W.R. MARTIN, "Boundary Projection Acceleration: A New Approach to Synthetic Acceleration of Transport Calculations," *Journal of Nuclear Science and Engineering*, **100**, 177-189 (1988).

[60] S. SANTANDREA and R. SANCHEZ, "Acceleration techniques for the characteristic method in unstructured meshes," *Annals of Nuclear Energy*, **29**, 323-352 (2002).

[61] K.S. SMITH, "Nodal Method Storage Reduction by Nonlinear Iteration," *Transactions of the American Nuclear Society*, **44**, 265 (1983).

[62] N.Z. CHO, "Fundamentals and Recent Developments of Reactor Physics Methods," *Journal of Nuclear Engineering and Technology*, **37**, 1, 25-78 (2005).

[63] H.J. KOPP, "Synthetic method solution for the transport equation," *Journal of Nuclear Science and Engineering*, **17**, 65 (1963).

[64] E.W. LARSEN and B.W. KELLEY, "CMFD and Coarse-Mesh DSA," *Proceedings of PHYSOR 2012 - Advances in Reactor Physics - Linking Research, Industry, and Education*, Knoxville, TN, USA, April 15-20 (2012), [CD-ROM].

[65] B.W. KELLEY and E.W. LARSEN, "CMFD Acceleration of Spatial Domain-Decomposed Neutron Transport Problems," *Proceedings of PHYSOR 2012 - Advances in Reactor Physics - Linking Research, Industry, and Education*, Knoxville, TN, USA, April 15-20 (2012), [CD-ROM].

[66] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, 2nd Edition, SIAM (2003).

[67] Y. SAAD, *Numerical Methods for Large Eigenvalue Problems*, Revised Edition, SIAM (2011).

[68] T. TAKEDA and H. IKEDA, "Final Report on the 3-D Neutron Transport Benchmarks," Tech. Rep. NEA/NEACRP/L(1990)330, Organisation for Economic Co-operation and Development Nuclear Energy Agency (1991).

[69] M.A. SMITH, E.E. LEWIS, and B.C. NA, "Benchmark on Deterministic Transport Calculations Without Spatial Homogenisation - A 2-D/3-D MOX Fuel Assembly Benchmark (C5G7 MOX Benchmark)," Tech. Rep. NEA/NSC/DOC(2003)16, Organisation for Economic Co-operation and Development Nuclear Energy Agency (2003).

[70] M.A. SMITH, E.E. LEWIS, and B.C. NA, "Benchmark on Deterministic Transport Calculations Without Spatial Homogenisation: MOX Fuel Assembly 3-D Extension Case," Tech. Rep. NEA/NSC/DOC(2005)16, Organisation for Economic Co-operation and Development Nuclear Energy Agency (2005).

[71] A.T. GODFREY, "VERA Core Physics Benchmark Progression Problem Specifications (Rev. 2)," Tech. Rep. CASL-U-2012-0131-002, Oak Ridge National Laboratory, http://www.casl.gov/publications.shtml (2013).

[72] J.Y. CHO and H.G. JOO, "Solution of the C5G7MOX benchmark three-dimensional extension problems by the DeCART direct whole core calculation code," *Progress in Nuclear Energy*, **48**, 456-466 (2006).

[73] R. FERRER, J.D. RHODES, and K.S. Smith, "Linear Source Approximation in CASMO5," *Proceedings of PHYSOR 2012 - Advances in Reactor Physics - Linking Research, Industry, and Education*, Knoxville, TN, USA, April 15-20 (2012), [CD-ROM].

[74] X.M. CHAI, K. WANG, and D. YAO, "The Linear Source Approximation in Three-Dimensional Characteristics Method," *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2009)*, Saratoga Springs, NY, USA, May 3-7 (2009), [CD-ROM].

[75] S. SANTANDREA, "An Integral Multidomain DP_N Operator as Acceleration Tool for the Method of Characteristics in Unstructured Meshes," *Journal of Nuclear Science and Engineering*, **155**, 223-235 (2007).

[76] Y.R. PARK and N.Z. CHO, "The MOC Neutron Transport Calculations Accelerated by Coarse-Mesh Angular Dependent Rebalance," *Proceedings of the Korean Nuclear Society*, Yongpyong, Korea, October (2004).

[77] W.R. BOYD, K.S. SMITH and B. FORGET, "A Massively Parallel Method of Characteristics Neutral Particle Transport Code for GPUs," *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2013)*, Sun Valley, ID, USA, May 5-9 (2013), [CD-ROM].

[78] Y. LIU, B. COLLINS, B. KOCHUNAS, W.R. MARTIN, K.S. KIM, and M.L. WILLIAMS, "Resonance Self-Shielding Methodology in MPACT," *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2013)*, Sun Valley, ID, USA, May 5-9 (2013), [CD-ROM].