Full Core 3D Neutron Transport Simulation Using the Method of Characteristics with Linear Sources

by

Geoffrey Alexander Gunow

B.S.E., University of Michigan (2012)

M.S., Massachusetts Institute of Technology (2015)

Submitted to the Department of Nuclear Science and Engineering
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computational Nuclear Science and Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2018

Author . . . .
Department of Nuclear Science and Engineering
March 1, 2018

Certified by . . . .
Kord Smith
KEPCO Professor of the Practice of Nuclear Science and Engineering
Thesis Supervisor

Certified by . . . .
Benoit Forget
Associate Professor of Nuclear Science and Engineering
Thesis Supervisor

Accepted by . . . .
Nicolas G. Hadjiconstantinou
Professor of Mechanical Engineering
Co-Director, Computational Science and Engineering Program

Accepted by . . . .
Ju Li
Battelle Energy Alliance Professor of Nuclear Science and Engineering
Professor of Materials Science and Engineering
Chair, Committee on Graduate Students

3D Method of Characteristics Simulation of Full-core Reactor Problems

by

Geoffrey A. Gunow

Submitted to the Department of Nuclear Science and Engineering
on March 1, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computational Nuclear Engineering

Abstract

The development of high fidelity multi-group neutron transport-based simulation tools for full core Light Water Reactor (LWR) analysis has been a long-standing goal of the reactor physics community. While direct transport simulations have previously been far too computationally expensive, advances in computer hardware have allowed large scale simulations to become feasible. Therefore, many have focused on developing full core neutron transport solvers that do not incorporate the approximations and assumptions of traditional nodal diffusion solvers.

Due to the computational expense of direct full core 3D deterministic neutron transport methods, many have focused on 2D/1D methods which solve 3D problems as a coupled system of radial and axial transport problems. However, the coupling of radial and axial problems also introduces approximations. Instead, the work in this thesis focuses on explicitly solving the 3D deterministic neutron transport equations with the Method of Characteristics (MOC).

MOC has been widely used for 2D lattice physics calculations due to its ability to accurately and efficiently simulate reactor physics problems with explicit geometric detail. The work in this thesis strives to overcome the significant computational cost of solving the 3D MOC equations by implementing efficient track generation, axially extruded ray tracing, Coarse Mesh Finite Difference (CMFD) acceleration, linear track-based source approximations, and scalable domain decomposition. Transport-corrected cross-sections are used to account for anisotropic without needing to store angular-dependent sources.

Additionally, significant attention has been given to complications that arise in full core simulations with transport-corrected cross-sections. The convergence behavior of transport methods is analyzed, leading to a new strategy for stabilizing the source iteration scheme for neutron transport simulations. The methods are incorporated into the OpenMOC reactor physics code and simulation results are presented for the full core BEAVRS LWR benchmark. Parameter refinement studies and comparisons with reference OpenMC Monte Carlo solutions show that converged full core 3D MOC simulations are feasible on modern supercomputers for the first time.

Thesis Supervisor: Kord Smith
Title: KEPCO Professor of the Practice of Nuclear Science and Engineering

Thesis Supervisor: Benoit Forget
Title: Associate Professor of Nuclear Science and Engineering

Acknowledgments

This thesis was supported in part by the Center for Exascale Simulation of Advanced Reactors (CESAR), a co-design center under the U.S. Department of Energy Contract No. DE-AC02-06CH11357. The author was also a recipient of the DOE Office of Nuclear Energy's Nuclear Energy University Programs Fellowship. The research was partially funded from the DOE Office of Nuclear Energy's Nuclear Energy University Programs (contract number: DE-NE0008578). This research made use of the resources of the High Performance Computing Center at Idaho National Laboratory, which is supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517. Additionally, this research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

I would like to thank my research advisors Kord Smith and Benoit Forget for their incredible support and expanding my knowledge of both nuclear science and computational science. The flexibility they gave me allowed me to pursue my academic interests in computing, greatly expanding both my knowledge and practical skills. Their guidance was critical during my time at MIT, pushing me to conquer larger goals. They were patient, supportive, and helpful when confronting difficult challenges throughout the research. I am incredibly grateful for their support and leadership and hope to maintain my collegial friendship with them as I embark on new challenges.

I am incredibly grateful for the support and guidance of my peers within the Computational Reactor Physics Group (CRPG). I am especially grateful to Sam Shaner and Will Boyd for their guidance in developing OpenMOC. Both were original developers of the OpenMOC code for 2D simulations and have helped me learn good software habits when developing OpenMOC. Sam Shaner also helped lay the groundwork for the 3D MOC solver presented in this thesis. Additionally, his work on track laydown algorithms was particularly helpful in implementing an efficient solver. Without his support, developing the 3D MOC solver have been tremendously difficult. In addition to his help in developing OpenMOC, Will Boyd's thesis on multi-group cross-section generation was directly used in this thesis for determining the input multi-group cross-sections for the BEAVRS benchmark. Both Sam Shaner and Will Boyd were extremely helpful when problems arose during the development of the 3D MOC solver in this thesis.

I would also like to thank my family: my parents Nancy and Sandy and my sister Genna. They have all been extremely supportive of my academic career and have been there for me every step of the way. From early grade school studies through graduate studies, they have always been encouraging. I would also like to thank my girlfriend Ankita for being there for me during the last difficult years of my graduate studies. She was incredibly patient during my long nights and weekends at the office, finishing my graduate work. She helped me remain focused on my work during the past years, allowing me to achieve my academic goals.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Definitions and Acronyms

API	Application Programming Interface
BEAVRS	Benchmark for Evaluation and Validation of Reactor Simulations
BP	Burnable Poison
CCM	Chord Classification Method
CSG	Constructive Solid Geometry
CMFD	Coarse Mesh Finite Difference
FLOP	Floating Point Operation
GMRES	Generalized Minimal Residual
HZP	Hot Zero Power
LWR	Light Water Reactor
MOC	Method of Characteristics
MPI	Message Passing Interface
MRT	Modular Ray Tracing
s-MRT	Simplified Modular Ray Tracing
PWR	Pressurized Water Reactor
RMS	Root Mean Square
SR	Source Region
SWIG	Simplified Wrapper Interface Generator

Chapter 1

Introduction

1.1 Motivation

Numerical simulation of neutron physics inside a nuclear reactor is fundamental to the design and operation of nuclear power plants. Neutron physics simulations are necessary for determining core reactivity, power distributions, isotopic depletion, and transient behavior. Accurate and predictive simulations can improve the operation of current nuclear reactors by reducing overly-conservative safety margins, incorporating accident tolerant fuels, and extending to longer operating cycle lengths. In addition, predictive neutron physics simulations are critical to the evaluation of advanced reactor designs, many of which operate in very different physics regimes than currently operating reactors.

Currently operating nuclear reactors often rely on nodal diffusion methods to simulate neutron physics. These methods are very fast and efficient, but have difficulty capturing localized details. In current generation nuclear reactor designs, such as the Westinghouse AP 1000™Pressurized Water Reactor (PWR), there is much greater geometric and material complexity than previous generations. These new complexities, such as axial and radial enrichment zoning and partial length Burnable Poisons (BPs), allow for greater efficiency and longer cycle lengths by reducing power peaking. However, the increase in axial and radial heterogeneity poses significant issues for nodal methods. Specifically, nodal diffusion solvers often assume smooth axial and radial variation of scalar flux distributions within fuel assemblies. Therefore, these complex features lead to axial and radial profiles which are not smooth enough for modern nodal methods to be accurate.

While higher-fidelity methods are capable of resolving local gradients and fine heterogeneous detail, they are often significantly slower. This can be prohibitive in reactor analysis where many simulations are required in a relatively short time frame. The goal of high-fidelity modeling in this realm is to create a tool that can benchmark and inform the development of nodal diffusion solvers. Even as a benchmark tool, high-fidelity neutron physics simulations can be too computationally intense to be useful. Therefore, there is a need for 3D high-fidelity neutron physics simulations that are accurate and reliable, but also computationally efficient. This thesis focuses on developing a high-fidelity 3D MOC neutron transport method capable of forming benchmark solutions in reasonable computational time.

1.2 Background

Nodal methods are the standard for full core neutron physics simulation in modern reactor analysis. In order to form reasonable solutions, these simulations rely on accurate multi-group cross-section data formed from a multi-level approach to decouple the energy, angular, and spatial dimensions as depicted in Figure 1-1 [1]. The multi-level approach typically combines high-fidelity models of energy self-shielding physics with low fidelity geometric models of unique core components. The energy complexity is then reduced as larger geometric models are considered.

The first stage of typical Light Water Reactor (LWR) multi-group cross-section generation attempts to capture self-shielding effects within an infinite array of fuel pins. This step typically condenses the continuous energy behavior of cross-sections to a ≈ 100 energy group cross-section set. The next stage captures spatial self-shielding effects between pins by using this cross-section set to simulate every unique fuel assembly assuming an infinite lattice and further reducing the group structure to a few group cross-section set. Lastly, these cross-sections are used in a full core nodal diffusion solver

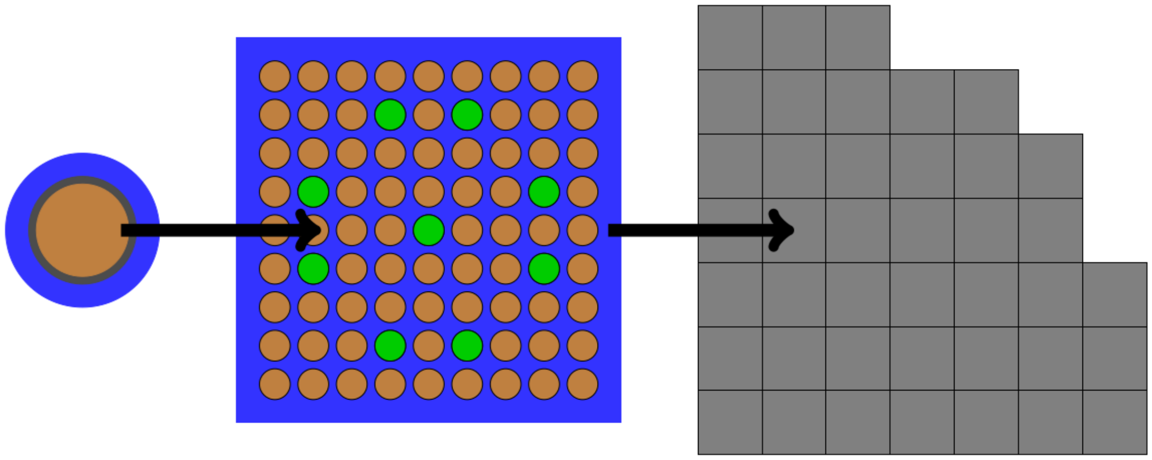


Figure 1-1: A depiction of the current multi-level simulation process for full core reactor analysis

where the neutron cross-sections over each assembly are homogenized. The resulting coarse mesh solution is then superimposed onto the individual single assembly solution to reconstruct local pin-powers.

Each step of the multi-level approach introduces assumptions and approximations. In order to develop a high-fidelity neutron transport simulation tool, these approximations should be removed. Monte Carlo simulations incorporate minimal approximations and are often viewed as the gold standard for accurate neutron physics simulation. However, an extremely large number of Monte Carlo histories is required to produce accurate pellet-level fission rates, leading to a very large computational cost.

An alternative approach can be taken whereby the full core is directly simulated using many-group transport methods. The multi-group cross-sections can be formed from direct Monte Carlo simulation of the full core problem where reaction rate tallies within regions are used to form accurate approximations of the multi-group cross-sections. Since these cross-section tallies can converge significantly faster than the Monte Carlo pin-powers using machine learning techniques [1], the computational cost of cross-section generation can be significantly reduced.

While the use of transport methods in place of nodal diffusion methods leads to a dramatic increase in computational requirements for reasonable solutions, recent advances in computational power for both engineering clusters and large scale supercomputers have enabled the computation of extremely large scale reactor simulations.

One such transport method is the Method of Characteristics (MOC), which discretizes the neutron transport equation using many characteristic neutron paths and directions which traverse the reactor geometry. MOC has seen widespread use for the 2D lattice physics analysis of single assemblies, as described previously in the multi-level approach. Additionally, 2D radial core slices of reactor geometries have been simulated using MOC. However, for reactors with significant axial heterogeneity, 3D methods should be used to capture the axial detail.

Due to the computational expense of direct full core 3D deterministic neutron transport methods, many have focused on 2D/1D methods [2–7] which solve 3D problems as a coupled system of radial and axial transport problems. A chosen number of radial slices are simulated with some 2D transport method, such as 2D MOC. Transverse leakage terms couple the radial problems through a 1D transport or diffusion method. While this approach reduces the computational burden in comparison with direct 3D transport simulations, it also introduces approximations. Instead of making such 2D/1D approximations, the work in this thesis focuses on explicitly solving the 3D deterministic neutron transport equations with MOC.

1.3 Literature Review of 3D MOC

Previously, others have attempted to simulate reactor physics problems using 3D MOC but have been limited to small models due to computational constraints of their particular 3D MOC implementations. Therefore, this thesis concentrates on efficiently solving the 3D MOC system of equations in order to make full core calculations feasible. Here, a variety of past and present 3D MOC implementations are discussed.

1.3.1 CACTUS

CACTUS was one of the first developed 2D MOC neutron transport simulators [8]. It is part of the WIMS core physics simulator and has been the standard lattice physics code used by the simulator for over 20 years. Recently, a new 3D MOC solver named CACTUS3D was added [9], reusing much of the CACTUS code. 3D geometry and ray tracing abilities were added for CACTUS3D, with much of the framework of the flux solver imported from CACTUS. Since CACTUS3D observed significant changes in track laydown for small changes in tracking parameters, a new 3D MOC solver was created as the main simulation tool in WIMS for solving core calculations [10]. This solver is named CACTUSOT since rays are treated in a "once-through" approach.

In the once-through scheme, each track assumes zero incoming angular flux and is only followed until it reaches the boundary of the geometry. Therefore this approach is limited to core problems with vacuum boundaries. Since tracks do not pass their angular fluxes to other tracks, as would be the case in problems with reflected boundaries, a cyclic track laydown with track linking at boundaries does not need to be enforced. Instead, tracks are generated to uniformly fill the geometry. This is accomplished by generating parallel tracks at every angle on each boundary surface, separated by constant spacing in all directions. This approach was largely adopted due to concerns over untracked elements and calculation of element volumes.

Both CACTUS3D and CACTUSOT explicitly save tracking data for segments. Whereas CACTUS3D stores tracking information for *every* segment, CACTUSOT uses a slice-based geometry treatment. In this approach, the geometry is split into different slices. Each slice represents the geometry over a certain axial interval. In common reactor problems there might only be a few unique slices. Tracks are generated on each slice rather than the full problem and the tracking data is only stored for each unique slice.

Since tracks are generated for each slice and are not guaranteed to align at slice interfaces, this creates an issue for determining connecting angular fluxes. This is overcome by using track cross-sectional area and outward-directed angular fluxes to compute leakage rates out of the slices on each radial mesh cell on the interface boundary. Inward-directed angular fluxes are then computed by using tack cross-sectional areas and the computed leakage rates.

CACTUSOT also implements features found in CACTUS including CMFD acceleration restricted to Cartesian mesh, treatments for both transport-corrected P0 and anisotropic P1 scatter, and a diamond difference representation of the neutron source along track segments. Parallelism is introduced with Message Passing Interface (MPI) in which work is decomposed by track direction.

1.3.2 DRAGON

DRAGON is a neutron transport code concentrating on lattice physics calculations for CANDU reactors. Since CANDU reactors contain fuel elements with reactivity control devices placed perpendicularly to the fuel channels, accurate calculations require full 3D treatment of the physics. Therefore, a 3D MOC solver named MCI [11] was implemented.

MCI solves the flat source MOC equations using a nested iterative process. In outer iterations, the fission source is updated along with the eigenvalue estimate. During inner iterations, the scattering source is updated and transport sweeps yield currents and fluxes. Outward currents are tallied at boundaries with an albedo boundary condition yielding new inward angular fluxes for the following inner iteration. Acceleration is provided with a self-collision rebalancing scheme in which the energy distribution is recalculated during every inner iteration.

A track merging technique was also implemented in which tracks crossing the same regions in the same order are merged. While this may be useful for simple geometries, there would not be many tracks that have the same crossings in complex geometries. Typical full core problems have many radial complexities, reducing the effectiveness of this scheme.

Distributed memory parallelism was also introduced into MCI [12] in which each process received a copy of all scalar fluxes and a group of tracks to handle. After completing each transport sweep, a global reduction communicates flux information through a sum operation on every region. Angular fluxes were also communicated through global broadcasts. In order to optimize load balancing, many partitioning schemes for the tracks across processes were tested. While the parallel efficiency was decent for the tested cases, the largest case only used 8 CPU cores. For the time of the tests, this was standard. However, for modern parallel computing, this number of cores is quite small, especially for large scale neutron transport calculations.

1.3.3 MOCFE

MOCFE is a finite element MOC code created at Argonne National Laboratory [13]. Using a finite element approach, the MOC system of equations with a flat source approximation are solved in matrix form. The iteration scheme consists of iterations over between-group and within-group components. In this nested iteration scheme, the inner iterations iteratively converged the within-group flux solution. Each inner iteration is a transport sweep. Though this scheme is not unusual, it does increase the number of transport sweeps needed to converge the problem.

MOCFE also does not link tracks at boundaries. Due to this implementation decision, approximations are required to treat reflective or periodic boundary conditions. However, the lack of track linking requirements allows for less constraints when constructing a quadrature set. MOCFE implements a quadrature set in which projections to spherical harmonics of the scattering kernel are exactly preserved. Tracks are also created symmetric in z with each track representing both forward and backward angular fluxes.

The parallelism in MOCFE is implemented with MPI [14] parallelization over trajectories such that each process receives a copy of all source regions. On every transport sweep, each process computes the variation of angular fluxes over its trajectories and accumulates their contribution to all source regions. In order to communicate the results, global reductions are required at the end of each transport sweep, which can be quite expensive.

Application of MOCFE to large scale problems distributed across many processors was studied at length [15]. In order to increase scalability, the entire space, angle, and energy variables were decomposed across MPI processes. While this does indeed increase the parallelism, it can lead to dramatically inefficient designs due to poor cache efficiency.

The decision to explicitly store MOC information in matrices can also lead to poor performance. Typical efficient MOC implementations implicitly compute matrix elements of the transport equations on-the-fly rather than storing the data in some typical matrix representation. MOCFE uses a Generalized Minimal Residual (GMRES) linear solver to solve the MOC equations. While GMRES can be quite efficient for arbitrary matrices, the MOC application is quite unique in which matrices can be easily inverted in a transport sweep due to the matrix structure. A GMRES implementation oblivious to this structure can incur significant overhead.

Since the computational demands of MOCFE were too large in comparison with other neutron transport simulation tools, the 3D MOC solver was abandoned when the code was merged into the PROTEUS neutron simulation suite. Instead of using a 3D MOC solver, the code was re-purposed to solve the transport equations using approximate 2D/3D methods in PROTEUS-MOC [16].

1.3.4 MMOC

Liu created a 3D MOC implementation based on modular ray tracing [17] named MMOC. His track laydown algorithm, which allows natural track linking over modular domains, is used in this thesis and thoroughly discussed in Chapter 4. This track laydown is efficient with only slight adjustments needed from desired user parameters, allowing for greatly reduced computational costs in comparison with other track laydown methods which are less flexible and require extra tracks to be inserted in order to ensure track linking at boundaries [18]. By considering entire 2D cycle lengths rather than individual 2D track lengths when forming track linking relationships, the requirements are significantly less stringent, allowing for the added flexibility.

In this thesis, the modular ray tracing algorithm for track generation is useful for linking tracks at domain boundaries during domain decomposition across many nodes. However, Liu used the modular tracking mainly to reduce ray tracing costs. MMOC approaches the MOC problem by splitting it into many modules. Over many of the modules, Liu notes that the geometry and track laydown are the same. Therefore, *typical cells* are identified and ray tracing is only computed for unique cells. For applications in this thesis, ray tracing costs can be trivial in comparison with the overall work of solving the MOC equations so the ray tracing aspect of MMOC is less applicable.

1.3.5 MPACT

Kochunas implemented a 3D MOC solver in MPACT [19, 20], the standard neutron transport code in the VERA core simulator [21]. This was the first 3D MOC solver implemented with the intention of solving full core PWR problems with full geometric detail. The solver used many of the common principles and structure of 2D MOC solvers at the time. For instance, it used a flat source approximation and explicitly stored segment information during a pre-processing ray tracing step at the beginning of the solver. Unlike other solvers, it used a strict source iteration scheme without any inner iterations.

The track laydown implemented in the 3D MOC solver relied on the Modular Ray Tracing scheme [17]. However, simplifications were made leading to the s-MRT method, discussed further in Chapter 4. One of the reasons presented for these simplifications was to avoid tracks intersecting corners. While the s-MRT method leads to a much simpler track laydown algorithm, it has serious drawbacks. Specifically, the axial ray spacing must be finer than the radial ray spacing. This is problematic for full core PWR problems in which there is much greater geometric detail in the radial direction than the axial direction. This can lead to an artificial increase in the number of tracks required to resolve typical PWR problems. Since the computational cost (both in memory usage and run time) scales linearly with the number of tracks, any large artificial increase in the number of tracks can significantly hinder the computational performance. Published MPACT results show only a modest artificial increase in the number of tracks, but a fine axial ray spacing was assumed to be necessary.

One of the benefits of using a modular ray tracing track laydown is the ability to naturally link tracks at domain boundaries when using spatial domain decomposition. The 3D MOC solver in MPACT implemented both spatial domain decomposition and angular domain decomposition using non-blocking MPI communication [14]. For spatial domain decomposition, the geometry is decomposed into N sub-domains where N is equal to the number of MPI processes per number of angular decompositions. An algorithm is implemented to subdivide the geometry in order to maximize the volume-to-surface area ratio of each sub-domain, reducing the relative communication costs which scale with surface area. For angular domain decomposition, each MPI process replicates the scalar fluxes of the associated sub-domain, using global reductions to sum all of the tallied contributions to local scalar fluxes across all processes. This reduction operation was shown to be a bottleneck in the parallel scalability of the algorithm.

During each transport sweep, the sweeping algorithm first loops over energy groups, then over angles within the angular sub-domain, then over all tracks within the angular sub-domain in parallel using OpenMP. In this shared memory parallelism implementation, each thread receives a full copy of the scalar fluxes within the sub-domain. This replication of information allows contention between threads to be reduced, but also greatly increases the memory footprint of storing scalar fluxes. In addition, a reduction operation is required to sum together all of the local thread scalar fluxes.

CMFD acceleration was implemented in MPACT using the GMRES linear solver in PETSc with a block ILU preconditioner to solve the associated linear system. The solver was domain decomposed spatially and a procedure was implemented to update angular fluxes at sub-domain boundaries [22].

Various benchmarks were evaluated for the MPACT 3D MOC solver. These benchmarks include the Takeda benchmark [23] and the C5G7 benchmark [24]. Unfortunately, the results did not seem to match well with the reference solution. To understand performance on realistic PWR problems, a realistic PWR assembly was constructed and modeled.

A theoretical computational performance model was developed, showing a very high computational cost for full resolving a full core PWR problem using the MPACT implementation of 3D MOC. No attempt was made to perform full core PWR simulations due to the exorbitant computing requirements.

1.3.6 APOLLO3

APOLLO3 is a nuclear reactor analysis code which recently incorporated a 3D MOC solver named TDT [25]. TDT solves the MOC equations in a nested iterative process, much like the MCI solver in DRAGON. However, TDT adds another layer of iterations. In the outer iterations, new fission sources are computed. Within each outer iteration, thermal iterations are performed in which the scattering source is recomputed. Within each thermal iteration, internal iterations are conducted in which the spatial distribution is resolved. These innermost iterations require computing transport sweeps at every step.

One of the main differences between TDT and other MOC solvers is the explicit treatment of boundaries. TDT classifies boundaries as open or closed. At open boundaries the angular flux is known. For example, at vacuum boundaries the incoming angular flux is known to be exactly zero. At closed boundaries, such as reflective or periodic boundaries, the angular flux is unknown. Whereas many solvers treat closed boundaries by estimating the angular flux as being the value from the previous iteration, TDT explicitly computes the angular flux at boundaries by following tracks from an open boundary (where the angular flux is known) along a connecting cycle until it reflects onto the track of interest.

This boundary treatment is of course not possible if the geometry is encompassed entirely by closed boundaries. In these cases, TDT guesses an initial angular flux and follows the track cycle until the initial guess becomes irrelevant. It is unclear whether this is only implemented for cases with all closed boundaries or if this technique is used more generally.

Another critical feature of TDT is the Chord Classification Method (CCM). In this method, storage requirements are reduced by characterizing segments (or chords) into different groups: horizontal, vertical, and mixed. For horizontal and vertical chords, their lengths can trivially be computed, and are referred to as *recognized chords*. In addition, all recognized chords of the same type (horizontal or vertical) of the same region will have the same length. Some mixed chords may become recognized chords under certain circumstances, but it is far more difficult than horizontal or vertical chords.

During an initial ray trace, the tracking information is compactly stored in Hit Surface Sequences, which only detail the type of intersections a track observes. These sequences, which only consist of one integer per segment or chord, can determine both the type of intersections encountered (recognized or unrecognized) as well as the regions which the segments overlap.

For all unrecognized chords, segment lengths as well as exponential terms are computed and explicitly stored. During transport sweeps, they are simply loaded from memory. For all recognized chords, their segment lengths and exponential terms are computed on-the-fly during the transport sweeps with no upfront storage of segment information. Before the transport sweep an interpolation table is stored for the exponential term. During the transport sweep, this table is used for the computation of exponential terms for recognized chords.

The effectiveness of the storage technique relies on many recognized chords. Many of the tests conducted on CCM have a high ratio of axial source height to axial ray spacing, therefore having many recognized chords, and yielding favorable computational results. However, if the axial ray spacing could be coarsened while maintaining solution accuracy, the number of potential recognized chords would be significantly lowered.

Initially, the TDT solver was dependent on a flat source approximation. Its accuracy was verified on a variety of benchmark problems [26, 27]. More recently, it has been extended to allow for axial polynomial expansions of the source [28]. The radial variation of the neutron source within each source region is still assumed to be flat. The authors cite radial heterogeneity and complexity for not using a higher order source approximation in the radial plane. The higher order source approximation requires significantly more computational work per segment but allows a much coarser axial mesh to be used. CCM is emphasized for its ability to mitigate the additional work required for the higher order source approximation as it allows for chords of equal lengths to be efficiently computed. While developed for arbitrary order axial polynomial expansions, the published results focus on quadratic axial sources.

The TDT implementation is parallelized with OpenMP. The work is divided in a task-based parallelism approach. Each thread receives a set of tracks and a full copy of the scalar flux accumulators. When a given thread changes angle, the local copy of scalar flux accumulators are synchronized with the global scalar flux accumulator. Since this synchronization can be a bottleneck for parallel scaling, tracks are sorted into groups to minimize the frequency of changing angle.

The TDT implementation has also recently extended its capabilities by adding DP_n synthetic acceleration which supports polynomial flux fields [29]. This is perhaps the first publication of explicitly treating higher order spatial source components with acceleration techniques.

1.3.7 The LEAF Method

A successor to the ASMOC3D method [30], the LEAF method [31,32] approaches the 3D MOC problem in a very unique manner. Instead of basing the method on characteristic lines, vertically oriented characteristic planes are used instead. These methods ray trace over certain axial zones, much like a 2D/1D method, assuming an axially extruded geometry in which every radial plane has the same meshing. The vertically oriented characteristic planes are chosen to link at interfaces – both between axial zones and between connecting planes along the same radial direction within the current axial zone. In the LEAF method, the angular flux at interfaces is represented with a 2nd order Legendre expansion.

The equations are cast in terms of collision probabilities, escape probabilities, and transmission probabilities for the planes. Numerical integration is applied for each characteristic plane in order to determine these probabilities. This numerical integration is accomplished by laying axially stacked tracks across the plane for a given polar angle. These tracks form the abscissa of the integration.

At this point, the method described is very similar to 3D MOC. Aside from the angular flux being represented at interfaces by a 2nd order Legendre expansion, the numerical integration by laying axially stacked tracks in the plane causes the method to be equivalent to filling the geometry full of characteristic lines. It is essentially a re-working of the equations to cast the problem in a different context.

However, the main difference for practical application comes when these probabilities are computed as part of a pre-processing step. Rather than explicitly computing the probabilities (collision, escape, and transmission) of all vertical planes in the problem, the probabilities are computed and tabulated for a range of conditions and interpolated during transport sweeps. The interpolation parameters are: polar angle, axial height of the vertical plane, radial thickness, and total cross-section. By creating the lookup tables only using the chosen abscissa, the amount of work can be greatly reduced.

Since the goal of this method is to treat coarse axial regions, higher order source approximations are introduced. Specifically, sources are formed up to 2nd order in the axial direction and first order in the radial direction. These higher order source approximations allow for a significantly coarsened mesh while maintaining accuracy, reducing the computational requirements of the method.

So far the method has only been tested on somewhat small problems. However, it shows great potential. As long as the required probabilities can be accurately interpolated from somewhat coarse abscissa and the 2nd order Legendre approximation of angular flux is sufficiently accurate, this method should outperform 3D MOC methods. However, if a very large number of abscissa are required or the 2nd order Legendre approximation is not sufficient, then explicit 3D MOC methods would be preferable.

1.4 Objective

This thesis seeks to develop an efficient 3D MOC solver which can solve large 3D reactor physics problems. The behavior of 3D MOC and the sensitivity of its solution to input parameters, such as mesh refinement, is studied on a variety of realistic reactor physics problems. However, the primary goal of this thesis is to directly use 3D MOC to accurately and efficiently simulate full core LWR problems. While simulation accuracy is highly dependent on input cross-section data, accuracy in this context refers to fully converging the problem in both space and angle for a cross-section set which allows for reasonable accuracy. The computational scale of full core LWR problems is extremely large with approximately 100 billion region-wise unknowns, assuming a 70 group cross-section library with a geometry defined by a 17×17 assembly lattice, each with $17 \times 17 \times 200$ pin-cells, and each pin-cell containing 24 source regions with one average scalar flux and three scalar flux moments for each group in each source region. Additionally, the angular space must be sufficiently covered for each region, leading to approximately 100 trillion angular-dependent unknowns for a segment density of ten thousand segments per source region.

1.5 Thesis Outline

This thesis starts with an introduction to the MOC method and derives its associated equations in Chapter 2. The traditional flat source approximation is discussed and a 3D track-based linear source approximation is introduced. Then, the implementation of the MOC algorithm in OpenMOC is discussed. An emphasis is placed on how the implementation aspects allow for computational efficiency. Chapter 3 introduces the software design of OpenMOC and how the internal structure was re-designed to accommodate 3D MOC methods studied in this thesis. Chapter 4 discusses the efficient modular track laydown which reduces the total number of tracks required to converge common reactor physics problems. Chapter 5 discusses the on-the-fly ray tracing used in OpenMOC to reduce the total memory footprint and increase cache efficiency. Chapter 6 discusses the efficient spatial domain decomposition introduced for both the MOC and CMFD solvers in OpenMOC.

With these components allowing for efficient computation, the convergence aspects of MOC are studied, illuminating issues with the traditional source iteration process with transport-corrected cross-sections. Significant discussion is given to the iteration process used to converge the equations from a linear algebra perspective. A diagonal stabilization fix is introduced in Chapter 7 which alleviates the convergence issues through damping of MOC flux updates. The effect of CMFD acceleration is also discussed.

Finally, simulation results of OpenMOC are presented on PWR geometries. Chapter 8 conducts sensitivity studies of MOC parameters on cut-outs of the BEAVRS benchmark. The optimal parameters are then used in Chapter 9 for full core simulations of the BEAVRS benchmark. Resulting fission distributions are compared with an OpenMC reference solution and slight sensitivity studies are also conducted for the full core in order to ensure the chosen MOC parameters sufficiently converge the full core fission distribution. The thesis ends with a summary of progress made in this research for full core deterministic transport methods in Chapter 10.

Chapter 2

The Method of Characteristics

Standard MOC solvers, such as the one developed and discussed in this thesis, solve the multi-group transport equations, relying on multi-group cross-sections, allowing the computation of reaction rates across any reactor geometry. This chapter discusses the MOC method, including both theoretical and practical aspects.

First, the multi-group transport equation is discussed. Then, the MOC equations are derived from the multi-group transport equation for both flat and linear source approximations. This chapter highlights the important equations for an MOC implementation and algorithmic considerations are noted.

2.1 The Multi-Group Transport Equation

The ultimate goal in neutron transport analysis is to determine the rates of neutron-induced reactions throughout the reactor core. Neutrons can undergo various reactions when they strike materials of the reactor core, often referred to as *target nuclei*. These interactions include scattering, capture, and fission, though others exist as well, and understanding neutron behavior is critical to determining the reaction rates.

The neutron population can be categorized by location $\mathbf{r}$, direction of travel $\mathbf{\Omega}$, and energy. In this thesis, multi-group transport is studied in which the continuous energy variable has been discretized into a number of energy groups. Reaction rates R_X^g of type X in energy group g can be calculated with multi-group cross-sections Σ_X^g and the *neutron angular flux* ψ_g as

$$R_X^g(\mathbf{r},\mathbf{\Omega}) = \Sigma_X^g(\mathbf{r},\mathbf{\Omega})\psi_g(\mathbf{r},\mathbf{\Omega}) \tag{2.1}$$

where the neutron angular flux ψ_g characterizes neutrons along a certain direction. Often, the cross-sections are independent of angle and integrated reaction rates over all angles are desired. This motivates the determination of *neutron scalar fluxes* $\phi_g(\mathbf{r})$, defined by

$$\phi_g(\mathbf{r}) = \int_{4\pi} d\mathbf{\Omega}\, \psi_g(\mathbf{r},\mathbf{\Omega}), \tag{2.2}$$

to often be the goal of neutron simulations. Cross-sections have a continuous energy dependence but reaction rates can be preserved with multi-group cross-sections through the use of the scalar flux. Specifically, the multi-group cross-sections Σ_X^g for group g with energy bounds E_g and E_{g-1} are formed using the continuous-energy scalar flux ϕ as

$$\Sigma_X^g(\mathbf{r}) = \frac{\int_{E_g}^{E_{g-1}} dE\, \Sigma_X(\mathbf{r},E)\phi(\mathbf{r},E)}{\int_{E_g}^{E_{g-1}} dE\, \phi(\mathbf{r},E)}. \tag{2.3}$$

It is important to note that the scalar flux is the solution of neutron transport simulation. Therefore, the correct scalar flux solution is necessary in order to form accurate cross-sections. Since such knowledge is not practical, approximations of the neutron flux are used to form multi-group cross-sections [1]. A depiction of this process is shown in Figure 2-1. Once multi-group cross-sections are formed, a balance of neutron losses and gains can yield the steady-state neutron transport equations. The relevant losses are net neutron leakage and total reaction rate with materials. The relevant neutron gains are neutron production from fission and in-scattering from other energy groups. This leads to the neutron transport equation [33–37] which describes the balance of loss and source terms in Eq. 2.4.

$$\begin{aligned}\mathbf{\Omega}\cdot\nabla\psi_g(\mathbf{r},\mathbf{\Omega}) + \Sigma_t^g(\mathbf{r},\mathbf{\Omega})\psi_g(\mathbf{r},\mathbf{\Omega}) &= \\ \frac{1}{4\pi}\left(\frac{\chi_g(\mathbf{r})}{k}\sum_{g'=1}^{G}\nu_{g'}(\mathbf{r})\Sigma_f^{g'}(\mathbf{r})\,\phi_{g'}(\mathbf{r}) + \sum_{g'=1}^{G}\Sigma_s^{g'\to g}(\mathbf{r})\,\phi_{g'}(\mathbf{r})\right)\end{aligned} \tag{2.4}$$

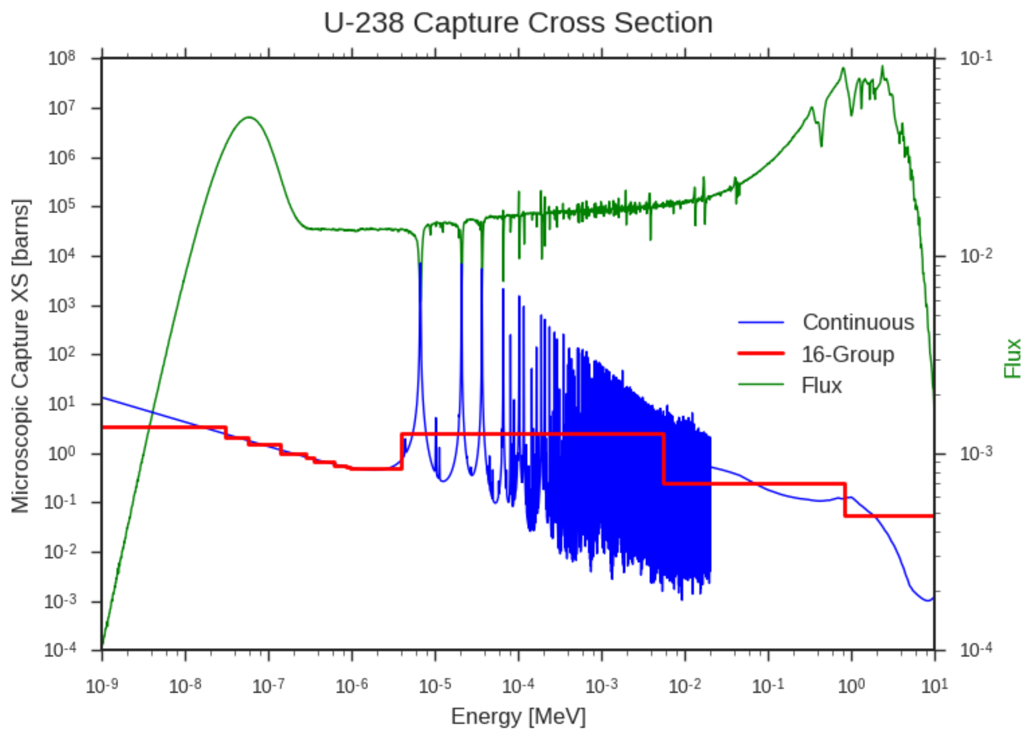


Figure 2-1: An illustration of forming multi-group cross-sections from continuous energy cross-sections showing the continuous energy capture cross-section of U-238 (blue), the neutron flux (green), and the multi-group representation with 16 groups (red).

where Σ_t^g is the total cross-section, χ_g is the neutron emission spectrum, ν is the average number of neutrons released per fission, Σ_f^g is the fission cross-section, $\Sigma_s^{g' \to g}$ is the scattering cross-section from group g' to group g, and the total number of energy groups is G. The eigenvalue k is applied to the fission term as one way of forcing a non-trivial solution when cross-sections are imperfectly known. If cross-sections and geometry were known to infinite precision for a steady-state system, k would be exactly 1.0. The deviation of k from 1.0 shows the degree to which the system is unbalanced for the supplied cross-sections.

It is important to note that the multi-group transport equation used here assumes isotropic scattering. However, neutron scattering off of light isotopes, such as hydrogen found in water, is highly anisotropic. Therefore, a *transport correction* [36] is often applied to the cross-sections to retain solution accuracy. This is discussed more thoroughly in Appendix F.3.

2.2 Derivation of Continuous Angle MOC Equations

Starting from the multi-group transport equation in Eq. 2.4, the neutron source $q_g(\mathbf{r})$ can be defined by Eq. 2.5 as

$$q_g(\mathbf{r}) = \frac{1}{4\pi}\left(\frac{\chi_g(\mathbf{r})}{k}\sum_{g'=1}^{G}\nu_{g'}(\mathbf{r})\Sigma_f^{g'}(\mathbf{r})\phi_{g'}(\mathbf{r}) + \sum_{g'=1}^{G}\Sigma_s^{g'\to g}(\mathbf{r})\phi_{g'}(\mathbf{r})\right) \tag{2.5}$$

which leads to the neutron balance expression,

$$\mathbf{\Omega}\cdot\nabla\psi_g(\mathbf{r},\mathbf{\Omega}) + \Sigma_t^g(\mathbf{r})\psi_g(\mathbf{r},\mathbf{\Omega}) = q_g(\mathbf{r}). \tag{2.6}$$

Next, a coordinate transformation is performed, casting the position as a displacement from an origin $\mathbf{r_0}$ along the direction of travel $\mathbf{\Omega}$ as $\mathbf{r} = \mathbf{r_0} + s\mathbf{\Omega}$. Under this transformation, the new balance equation becomes:

$$\mathbf{\Omega}\cdot\nabla\psi_g(\mathbf{r_0}+s\mathbf{\Omega},\mathbf{\Omega}) + \Sigma_t^g(\mathbf{r_0}+s\mathbf{\Omega})\psi_g(\mathbf{r_0}+s\mathbf{\Omega},\mathbf{\Omega}) = q_g(\mathbf{r_0}+s\mathbf{\Omega}) \tag{2.7}$$

In this form, the gradient reduces to a simple derivative by the distance s traveled from the origin $\mathbf{r_0}$ as

$$\frac{d\psi_g(\mathbf{r_0}+s\mathbf{\Omega},\mathbf{\Omega})}{ds} + \Sigma_t^g(\mathbf{r_0}+s\mathbf{\Omega})\psi(\mathbf{r_0}+s\mathbf{\Omega},\mathbf{\Omega}) = q_g(\mathbf{r_0}+s\mathbf{\Omega}). \tag{2.8}$$

Considering a region i with constant total cross-section $\Sigma_t^{i,g}$, the angular flux for a distance s from the origin can be evaluated analytically as

$$\psi_g(\mathbf{r_0}+s\mathbf{\Omega},\mathbf{\Omega}) = \psi_g(\mathbf{r_0},\mathbf{\Omega})e^{-\Sigma_t^{i,g}s} + \int_0^s ds'\, e^{-\Sigma_t^{i,g}(s-s')}q_g(\mathbf{r_0}+s'\mathbf{\Omega}). \tag{2.9}$$

Eq. 2.9 reveals an important relationship. For any region of constant total cross-section with known source distribution $q_g(\mathbf{r})$ and angular flux at a single point $\psi_g(\mathbf{r_0},\mathbf{\Omega})$, the angular flux for all points along the direction of travel $\mathbf{\Omega}$ can be calculated. Therefore, knowledge of incoming angular fluxes on the surface of any enclosed boundary is sufficient for the calculation of all angular fluxes within the region.

If the problem domain can be represented (or approximated) as the composition of a finite number of *source regions* over which the total cross-section is constant and the neutron source $q_g(\mathbf{r})$ takes some known distribution, the calculation of all angular fluxes throughout the problem is straightforward.

Realistically, the source distribution is not known before solving the neutron transport equation, since it depends on the scalar fluxes. However, if the geometry is sufficiently discretized, a low-order approximation of the *shape* of the neutron source within each region can be made with little impact on solution accuracy. An example of geometry discretization is shown in Fig. 2-2 where a fuel pin-cell is discretized radially. The image on the left shows a radial view of the fuel pin-cell geometry, colored by (constant cross-section) material region. The image on the right shows a discretized geometry, colored by source region over which the neutron source is assumed to have some low-order form.

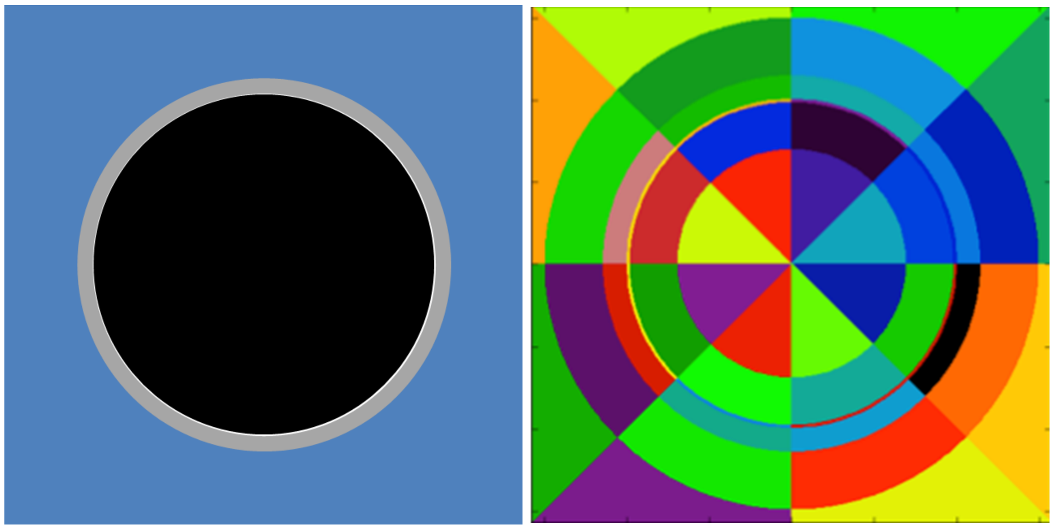

(a) **(b)**

Figure 2-2: A description of source region discretization and mesh refinement. The geometry is shown (a) colored by material and (b) colored by source region.

2.3 Track Discretization of the MOC Equations

MOC discretizes the angular space by choosing a finite number of directions to lay down *tracks* across the geometry. For each direction, tracks span the entire geometry between outer boundaries. A collection of tracks is termed the *track laydown*. A radial view of a coarse track laydown for a simple pin-cell geometry is shown in Figure 2-3. The geometry is colored by material region; namely water, clad, gap, and fuel. Note that the final track laydown traverses tracks forward and backward, rather than generating tracks in each direction, for computational efficiency [38].

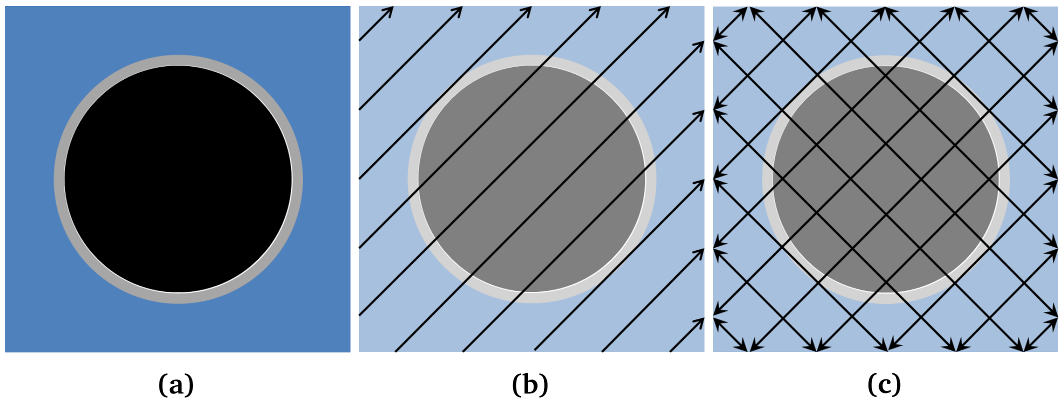


Figure 2-3: A description of the track laydown process. The geometry (a) is shown followed by the track laydown (b) for one particular direction. Finally the entire track laydown for 4 azimuthal angles is shown (c) with arrows showing that each track is traversed both forward and backward. Note that the track laydowns shown here are significantly coarser than usual track laydowns for illustration purposes.

For each track crossing a given constant cross-section region, the variation of angular flux follows Eq. 2.9. Therefore, each track is discretized into *segments*, in which each segment is the portion of the track that crosses a particular region. Across each segment, Eq. 2.9 applies. An illustration of the segmentation process for the simple pin-cell geometry is shown in Figure 2-4. The presented track, with the geometry drawn along its direction of travel is discretized into segments by material region. For simplicity, this illustration does not discretize the source regions further than material boundaries. A real segmentation process might discretize segments over the domain shown in Figure 2-2b. This process would treat all tracks shown in the track laydown (eg. Figure 2-3) in a similar manner. These images show track laydown and segmentation in just the radial plane, but the same methodology applies to three dimensional geometries.

Now that tracks have been segmented, a system of equations can be formed governing the variation of angular flux through each region as presented in Eq. 2.9. However, this expression has a dependence on the incoming angular flux. For segments originating in the bulk of the geometry, continuity of angular flux is naturally enforced. The outgoing flux of the preceding segment in the track provides the incoming flux of the segment.

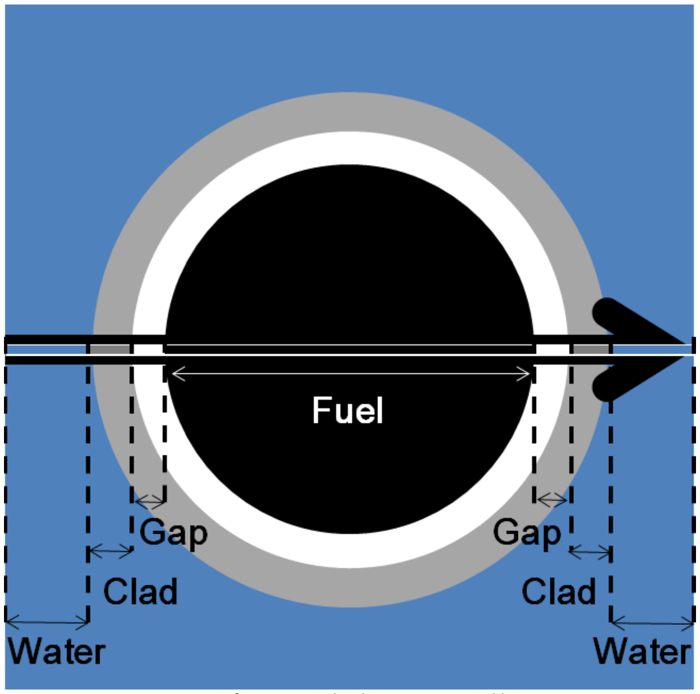


Figure 2-4: The segmentation of a track horizontally traversing a pin-cell domain with coarse source discretization wherein the boundaries of source regions are no finer than their material region. The formed segments are colored by material region. For illustration purposes, the geometry is not shown to scale.

Specifically, the position-dependent angular flux $\psi_{p,g}^{t,\varsigma}(\mathbf{r})$ in group g along track t and segment ς can be expressed as the angular flux of the preceding segment $\varsigma - 1$ at the interface point $\mathbf{r}_{\text{int}}$ where the segments meet as

$$\psi_{p,g}^{t,\varsigma}(\mathbf{r}_{\text{int}}) = \psi_{p,g}^{t,\varsigma-1}(\mathbf{r}_{\text{int}}). \tag{2.10}$$

Defining the angular flux along a segment in terms of distance traveled along the segment rather than position, this can equivalently be written as

$$\psi_{g}^{t,\varsigma}(0) = \psi_{g}^{t,\varsigma-1}(\ell_{t,\varsigma-1}) \tag{2.11}$$

where $\ell_{t,\varsigma}$ refers to the length of segment ς along track t. For angular fluxes originating at the geometry boundary, the boundary condition provides the relationship for the angular flux.

For reflective boundary conditions, the incoming angular flux is equal to the outgoing angular flux of the reflected angle. This requires that a track be present with the correct reflecting angle, meeting at precisely the same point on the boundary. This is enforced during the track laydown, further discussed in Chapter 4.

Tracks are defined to originate at one boundary and span the geometry until they terminate at another boundary. The segments of a track are ordered in the direction of the track so that only the first segment is affected by a boundary condition. Track t is defined to have $S(t)$ segments. For vacuum boundaries,

$$\psi_g^{t,1}(0) = 0. \tag{2.12}$$

For boundary conditions (such as reflective, periodic, and rotational) that connect with another track, the function C is defined which yields the linking track. For instance, the incoming angular flux of track t would connect with the outgoing angular flux of track $C(t)$ as

$$\psi_g^{t,1}(0) = \psi_g^{C(t),S(C(t))}(\ell_{C(t),S(C(t))}) \tag{2.13}$$

With the angular and spatial discretization from the track laydown, the angular flux relationship along a characteristic path given in Eq. 2.9 can be presented in terms of the discretized track segments in Eq. 2.14 where i is the region traversed by track t and segment ς.

$$\psi_g^{t,\varsigma}(s) = \psi_g^{t,\varsigma}(0)e^{-\Sigma_t^{i,g}s} + \int_0^s ds'\, e^{-\Sigma_t^{i,g}(s-s')} q_{t,\varsigma,g}(s) \tag{2.14}$$

Note that when the transformation to distance s is performed on the source, it also becomes dependent on the track segment even though the source is assumed to be isotropic. This is because the source can have a shape within the region and track segments traverse the region at different positions.

Still, these equations only determine the behavior of the neutron flux along a particular track. However, the region-averaged scalar flux is often desired. The average scalar flux $\overline{\phi_{i,g}}$ in a region i for group g can be computed by integrating over all directions as

$$\overline{\phi_{i,g}} = \frac{1}{V_i} \int_V dV \int_{4\pi} d\Omega\, \psi_g(\mathbf{r}, \mathbf{\Omega}). \tag{2.15}$$

In order to numerically calculate the integral, each track represents a volume formed by the product of its length and both the radial and axial perpendicular distances to tracks of the same direction. Those perpendicular lengths form the track cross-sectional area as illustrated in Figure 2-5. This illustration just shows the radial view of the

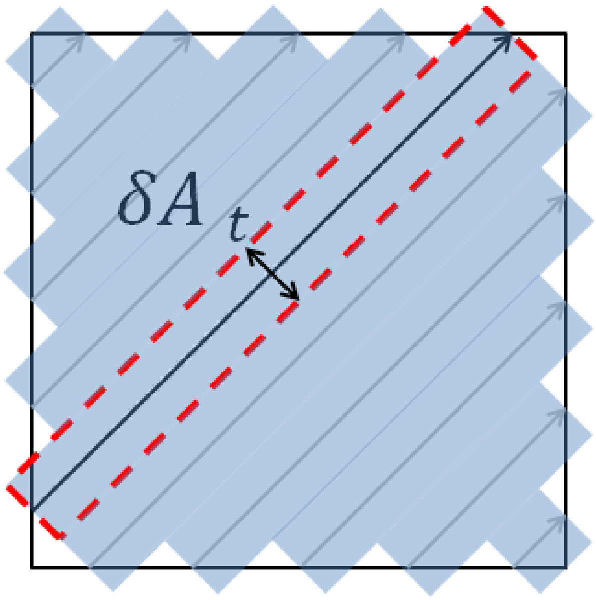


Figure 2-5: An illustration of the spatial volume represented by each track as the product of track length and track cross-sectional area δA_t. The volume represented by each track is shaded in blue.

tracks. However, in three dimensions, axial distances also exist between tracks, so the track cross-sectional area does indeed represent an area rather than a distance. With a fine track laydown, the discretization error becomes small. Similarly, each track t has an angular weight α_t relating to the width of the angular space represented by the track. Therefore the overall weight of the track w_t is calculated as the product of track cross-sectional area and angular weight as given in Eq. 2.16.

$$w_t = \delta A_t \alpha_t \tag{2.16}$$

With each track representing a discretized portion of the spatial and angular domain, Eq. 2.15 can be transformed to reflect the track discretization in Eq. 2.17, yielding a closed-form relationship to calculate the scalar flux.

$$\overline{\phi_{i,g}} = \frac{1}{V_i} \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds\, \psi_g^{t,\varsigma}(s) \tag{2.17}$$

2.4 Track Simplifications and Calculation of Volumes

In the previous section, the track discretization of the MOC equation was introduced. In the section, it was noted that tracks are used in both forward and backward directions. In the notation, a given track t represents only a single direction. Due to the two directional tracks traversing the same line, simplifications can sometimes be made to the equations.

Consider the weighted summation of values $r_{t,\varsigma}$ which are potentially dependent on the traversed segment ς of track t. The summation can be cast as

$$\sum_{(t,\varsigma)\in V_i} w_t r_{t,\varsigma} = \sum_{(t,\varsigma)\in V_i^F} w_t r_{t,\varsigma} + \sum_{(t,\varsigma)\in V_i^B} w_t r_{t,\varsigma} \tag{2.18}$$

where V_i^F refers to the volume tracked by forward traversed tracks and V_i^B refers to the volume tracked by backward traversed tracks. Note that the corresponding forward/backward tracks have opposite directions. Therefore, if quantity $\tilde{r}_{t,\varsigma}$ does not depend on the direction $\mathbf{\Omega}$, then for any direction $v \in (x, y, z)$ and any *odd* integer d,

$$\sum_{(t,\varsigma)\in V_i} w_t \left(\Omega_{v,t}\right)^d \tilde{r}_{t,\varsigma} = 0. \tag{2.19}$$

Similarly, for any *even* integer d,

$$\sum_{(t,\varsigma)\in V_i} w_t \left(\Omega_{v,t}\right)^d \tilde{r}_{t,\varsigma} = 2 \sum_{(t,\varsigma)\in V_i^F} \left(\Omega_{v,t}\right)^d w_t \tilde{r}_{t,\varsigma}. \tag{2.20}$$

The calculation of region volumes is one example where this simplification can be used. In order for the MOC equations to be accurate, the condition

$$\frac{1}{V_i} \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds\, 1 = 1 \tag{2.21}$$

should apply. However, for a coarse track laydown the volume is not perfectly accurate, causing the condition not to be met. Therefore, volumes are calculated with the tracked lengths and weights as

$$V_i = \sum_{(t,\varsigma)\in V_i} w_t \ell_{t,\varsigma} \tag{2.22}$$

which forces the aforementioned condition to be met. This can be simplified by noting that $\ell_{t,\varsigma}$ is not dependent on direction, allowing the volume to be calculated only using forward-directed tracks as

$$V_i = 2 \sum_{(t,\varsigma)\in V_i^F} w_t \ell_{t,\varsigma}. \tag{2.23}$$

2.5 Flat Source Approximation

One common low-order approximation for the neutron source is the flat source approximation. This approximation assumes the neutron source for a given energy group $q_g(\mathbf{r})$ is spatially constant over each source region. The flat source approximation is also the most frequently used approximation in standard MOC solvers [9,11,17,20,27,39,40]. For this approximation, the angular flux relationship given in Eq. 2.14 reduces to

$$\psi_g^{t,\varsigma}(s) = \psi_g^{t,\varsigma}(0)e^{-\Sigma_t^{i,g}s} + \frac{q_{i,g}^0}{\Sigma_t^{i,g}} F_1\left(\Sigma_t^{i,g}s\right) \tag{2.24}$$

with region i having constant neutron source $q_{i,g}^0$ where

$$F_1(\tau) = 1 - e^{-\tau}. \tag{2.25}$$

This equation allows for the computation of all angular fluxes within a region of neutron source $q_{i,g}^0$. From the definition of the neutron source in Eq. 2.5, the constant neutron source $q_{i,g}^0$ in the region i can be computed as

$$q_{i,g}^0 = \frac{1}{4\pi}\left(\frac{\chi_{i,g}}{k}\sum_{g'=1}^{G} \nu_{i,g'}\Sigma_f^{i,g'}\overline{\phi_{i,g'}} + \sum_{g'=1}^{G}\Sigma_s^{i,g'\to g}\overline{\phi_{i,g'}}\right) \tag{2.26}$$

where $\overline{\phi_{i,g}}$ is the average scalar flux in the region and the cross-sections have been taken to be constant over each region i. Combining Eq. 2.17 with Eq. 2.24, the scalar flux can be calculated using the relationship in Eq. 2.27.

$$\overline{\phi_{i,g}} = \frac{q_{i,g}^0}{\Sigma_t^{i,g}} + \frac{1}{\Sigma_t^{i,g}V_i}\sum_{(t,\varsigma)\in V_i} w_t\left(\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma})\right) \tag{2.27}$$

The calculated fluxes can then be used to construct the neutron sources $q^0_{i,g}$ for each region i and each energy group g with the relationship in Eq. 2.26. The solution of the MOC system of equations is discussed in great detail in Appendix A. It is important to note that the application of MOC equations over all segments (Eq. 2.17 and Eq. 2.24) for a given source distribution is termed a *transport sweep*.

2.6 Track-based Linear Source Approximation

In the previous section, the MOC equations were derived using a flat source approximation. While this approximation is convenient and used by many, a linear approximation can potentially reduce the computational requirements of simulating a spatially converged reactor problem. While the linear source approximation increases the computational cost for a fixed discretization, the higher-order source can capture source gradients, allowing for a much coarser mesh discretization while maintaining solution accuracy [41]. In this section, the track-based linear source approximation developed by Ferrer [42] for 2D MOC is extended to 3D MOC. A discussion of the subtleties in the derivation and implementation is presented.

2.6.1 Derivation of the Linear Source Approximation

With a linear source approximation, the source is assumed to vary linearly over a given track t on segment ς that traverses region i as

$$q_{t,\varsigma,g}(s) = q^0_{t,\varsigma,g} + q^1_{t,\varsigma,g}(s - \ell_{t,\varsigma}/2) \tag{2.28}$$

where $\ell_{t,\varsigma}$ is the length of the track t over the region i and the track-dependent coefficients are $q^0_{t,\varsigma,g}$ and $q^1_{t,\varsigma,g}$. It is important to note that in this definition the source components are dependent on the track. Later, the track description of the source will be transformed into one dependent on only the source region, not on the track. For a linear source defined by the spatial region, the source for each track varies since tracks enter the source regions at different positions and different angles. This motivates track-dependent source components.

By inserting the linear source definition into the balance equation, the angular flux follows the relationship

$$\psi_g^{t,\varsigma}(s) = \psi_g^{t,\varsigma}(0)e^{-\Sigma_t^{i,g}s} + \int_0^s ds' \left(q_{t,\varsigma,g}^0 + q_{t,\varsigma,g}^1(s' - \ell_{t,\varsigma}/2)\right)e^{-\Sigma_t^{i,g}(s-s')}. \tag{2.29}$$

In this form, it is possible to analytically calculate the angular flux variation along any track given the track-based linear source components. In the following subsections, the linear source equations will be derived starting from this equation and the definition of the linear source.

2.6.1.1 Calculation of Average Scalar and Angular Fluxes

The calculation of average scalar fluxes is often the goal of neutron transport simulations. Therefore, this derivation of the track-based linear source approximation starts with a discussion of their calculation given the linear source definition. The angular flux relationship previously presented in Eq. 2.29 can be simplified to

$$\psi_g^{t,\varsigma}(s) = \psi_g^{t,\varsigma}(0) + \left(\frac{q_{t,\varsigma,g}^0}{\Sigma_t^{i,g}} - \psi_g^{t,\varsigma}(0)\right)F_1\left(\Sigma_t^{i,g}s\right) + \left(\frac{q_{t,\varsigma,g}^1}{2\left(\Sigma_t^{i,g}\right)^2}\right)F_2\left(\Sigma_t^{i,g}s, \Sigma_t^{i,g}\ell_{t,\varsigma}\right) \tag{2.30}$$

where the function F_1 follows the form given in Eq. 2.25 and F_2 is defined in terms of F_1 as

$$F_2(\tau_1, \tau_2) = 2\left[\tau_1 - F_1(\tau_1)\right] - \tau_2 F_1(\tau_1) \tag{2.31}$$

Recall that the average flux can be computed by integrating angular fluxes as presented in Eq. 2.17. This can be re-written in terms of average angular fluxes as

$$\overline{\phi_{i,g}} = \frac{1}{V_i}\sum_{(t,\varsigma)\in V_i} w_t \overline{\psi}_g^{t,\varsigma}\ell_{t,\varsigma} \tag{2.32}$$

where the angular average angular flux $\overline{\psi}_g^{t,\varsigma}$ over a segment is defined by

$$\overline{\psi}_g^{t,\varsigma} = \frac{1}{\ell_{t,\varsigma}} \int_0^{\ell_{t,\varsigma}} ds\, \psi_g^{t,\varsigma}(s). \tag{2.33}$$

After the MOC transform and with the linear source approximation, the neutron transport equation takes the form

$$\frac{d\psi_{i,g}(s)}{ds} + \Sigma_t^{i,g}\psi(s) = q^0_{t,\varsigma,g} + q^1_{t,\varsigma,g}(s - \ell_{t,\varsigma}/2) \tag{2.34}$$

when the angular and spatial discretization is applied. This relationship can be rearranged to solve for the average angular flux $\overline{\psi}_g^{t,\varsigma}$ as

$$\overline{\psi}_g^{t,\varsigma} = \frac{q^0_{t,\varsigma,g}}{\Sigma_t^{i,g}} + \frac{\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma})}{\Sigma_t^{i,g}\ell_{t,\varsigma}} \tag{2.35}$$

From a neutron source definition presented in Eq. 2.28, the angular fluxes and their segment averages can be calculated using Eq. 2.35 with Eq. 2.30, respectively. The scalar fluxes can also be computed by combining Eq. 2.32 with Eq. 2.35, yielding

$$\overline{\phi_{i,g}} = \frac{1}{V_i} \sum_{(t,\varsigma)\in V_i} w_t \ell_{t,\varsigma} \left(\frac{q^0_{t,\varsigma,g}}{\Sigma_t^{i,g}} + \frac{\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma})}{\Sigma_t^{i,g}\ell_{t,\varsigma}} \right) \tag{2.36}$$

which can be simplified as

$$\overline{\phi_{i,g}} = \frac{\overline{q}_{i,g}}{\Sigma_t^{i,g}} + \frac{1}{\Sigma_t^{i,g} V_i} \sum_{(t,\varsigma)\in V_i} w_t \left(\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma}) \right) \tag{2.37}$$

which is the same analytic form as derived from the flat source approximation in Eq. 2.27.

2.6.1.2 Linear Source Defined By Region

Until now, the linear source approximation has been introduced in the perspective of each track. This is convenient for simplifying the MOC equations, but not convenient for computing region-dependent components. Therefore, a region-wise linear source q_i^g

for region i and energy group g is defined in terms of position $\mathbf{r}$ as

$$q_{i,g}(\mathbf{r}) = \overline{q}_{i,g} + \vec{q}_{i,g} \cdot \left(\mathbf{r} - \mathbf{r}_i^C\right) \tag{2.38}$$

where $\overline{q}_{i,g}$ is the flat source component, $\vec{q}_{i,g}$ is a vector representing the gradient of the source, and $\mathbf{r}_i^C$ is the centroid of the region i. The source gradient can be defined in terms of its components as $\vec{q}_{i,g} = \left[q_{x,i,g}, q_{y,i,g}, q_{z,i,g}\right]^T$ where $q_{x,i,g}$, $q_{y,i,g}$, and $q_{z,i,g}$ are the source gradients in the x, y, and z directions, respectively. Here the vector notation $\vec{q}$ was chosen to avoid confusion with $\mathbf{q}$, the vector of all sources across all regions and energy groups. With this formalism, the track-based linear source components defined in Eq. 2.28 can be computed as:

$$q^0_{t,\varsigma,g} = q_{i,g}(\mathbf{r}^m_{t,\varsigma}) = \overline{q}_{i,g} + \vec{q}_{i,g} \cdot \left(\mathbf{r}^m_{t,\varsigma} - \mathbf{r}_i^C\right) \tag{2.39}$$

$$q^1_{t,\varsigma,g} = \vec{q}_{i,g} \cdot \mathbf{\Omega}_t \tag{2.40}$$

where $\mathbf{r}^m_{t,\varsigma}$ is the midpoint of segment ς along track t and $\mathbf{\Omega}_t$ is its unit vector direction. In Cartesian coordinates, the source defined in Eq. 2.38 can be defined as

$$q_{i,g}(x,y,z) = \overline{q}_{i,g} + q_{x,i,g}\left(x - x_i^C\right) + q_{y,i,g}\left(y - y_i^C\right) + q_{z,i,g}\left(z - z_i^C\right) \tag{2.41}$$

where x_i^C, y_i^C, and z_i^C represent the x, y, and z coordinates of the region i centroid, respectively. The centroid represents the midpoint of the cell, as observed by traversing segments. The source can be written more compactly as

$$q_{i,g}(x,y,z) = \overline{q}_{i,g} + \sum_{v \in (x,y,z)} q_{v,i,g}\left(v - v_i^C\right) \tag{2.42}$$

where v represents the spatial variables x, y, and z. Using this notation, the track-based source components can be expressed as:

$$q^0_{t,\varsigma,g} = \overline{q}_{i,g} + \sum_{v \in (x,y,z)} q_{v,i,g}\left(v^m_{t,\varsigma} - v_i^C\right) \tag{2.43}$$

$$q^1_{t,\varsigma,g} = \sum_{v \in (x,y,z)} \Omega_{v,t} q_{v,i,g} \tag{2.44}$$

where $\Omega_{v,t}$ refers to the $v \in (x, y, z)$ component of the unit vector direction $\boldsymbol{\Omega}_t$ for track t and $v^m_{t,\varsigma}$ refers to the v coordinate of the midpoint of segment ς along track t. The v coordinate of the centroid, v^C_i, can be computed as

$$v^C_i = \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds\, v \qquad \forall v \in (x, y, z). \tag{2.45}$$

The variables x, y, and z corresponding to v can be cast in terms of the distance s along a segment as

$$v = v^m_{t,\varsigma} + \Omega_{v,t}\left(s - \frac{\ell_{t,\varsigma}}{2}\right) \qquad \forall v \in (x, y, z) \tag{2.46}$$

which can be inserted in Eq. 2.45 to yield the simplified form as

$$v^C_i = \sum_{(t,\varsigma)\in V_i} w_t \ell_{t,\varsigma} v^m_{t,\varsigma} \qquad \forall v \in (x, y, z). \tag{2.47}$$

The flat source component $\overline{q}_{i,g}$ can be computed using the scalar fluxes in the same way as the flat source approximation,

$$\overline{q}_{i,g} = \frac{1}{4\pi}\left(\frac{\chi_{i,g}}{k}\sum_{g'=1}^{G} \nu_{i,g'}\Sigma^{i,g'}_f \overline{\phi_{i,g'}} + \sum_{g'=1}^{G} \Sigma^{i,g'\to g}_s \overline{\phi_{i,g'}}\right). \tag{2.48}$$

2.6.1.3 Relating Linear Source Components To Moments

With the neutron source defined in terms of spatial variables rather than track-based variables, it is possible to relate the linear source components to source moments. The moment $Q_{v,i,g}$ of the source in the $v \in (x, y, z)$ direction can be calculated as

$$Q_{v,i,g} = \int_{V_i} d\mathbf{r}\,\left(v - v^C_i\right) q(\mathbf{r}) \qquad \forall v \in (x, y, z) \tag{2.49}$$

where V_i is the volume of region i. With the track discretization, the integral can be calculated as

$$Q_{v,i,g} = \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds \left(v - v_i^C\right)\left(\overline{q}_{i,g} + \sum_{v'\in(x,y,z)} q_{v',i,g}\left(v' - v'^C_i\right)\right) \qquad \forall v \in (x,y,z). \tag{2.50}$$

This can be separated as

$$\begin{aligned} Q_{v,i,g} = &\sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds \left(v - v_i^C\right)\overline{q}_{i,g} + \\ &\sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds \left(v - v_i^C\right)\left(\sum_{v'\in(x,y,z)} q_{v',i,g}\left(v' - v'^C_i\right)\right) \quad \forall v \in (x,y,z) \end{aligned} \tag{2.51}$$

and simplified to

$$Q_{v,i,g} = \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds \left(v - v_i^C\right)\left(\sum_{v'\in(x,y,z)} q_{v',i,g}\left(v' - v'^C_i\right)\right) \qquad \forall v \in (x,y,z) \tag{2.52}$$

due to the definition of the region centroid in Eq. 2.45. Defining moment coefficients $M_{v,v'}$ such that

$$M_{i,v,v'} = \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds \left(v - v_i^C\right)\left(v' - v'^C_i\right) \qquad \forall (v,v') \in (x,y,z)\times(x,y,z), \tag{2.53}$$

it becomes clear that this can be cast as a linear problem such that

$$M_i \vec{q}_{i,g} = \vec{Q}_{i,g} \tag{2.54}$$

where $\vec{Q}_{i,g} = \left[Q_{x,i,g}, Q_{y,i,g}, Q_{z,i,g}\right]^T$ and the matrix M_i is defined by Eq. 2.53 where

$$M_i = \begin{bmatrix} M_{i,xx} & M_{i,xy} & M_{i,xz} \\ M_{i,xy} & M_{i,yy} & M_{i,yz} \\ M_{i,xz} & M_{i,yz} & M_{i,zz} \end{bmatrix}. \tag{2.55}$$

Altogether, the linear system is defined by

$$\begin{bmatrix} M_{i,xx} & M_{i,xy} & M_{i,xz} \\ M_{i,xy} & M_{i,yy} & M_{i,yz} \\ M_{i,xz} & M_{i,yz} & M_{i,zz} \end{bmatrix} \begin{bmatrix} q_{x,i,g} \\ q_{y,i,g} \\ q_{z,i,g} \end{bmatrix} = \begin{bmatrix} Q_{x,i,g} \\ Q_{y,i,g} \\ Q_{z,i,g} \end{bmatrix}. \qquad (2.56)$$

The source moments can then be calculated similarly to the flat source components, using the relationship

$$Q_{\nu,i,g} = \frac{1}{4\pi}\left(\frac{\chi_{i,g}}{k}\sum_{g'=1}^{G} \nu_{i,g'}\Sigma_f^{i,g'}\hat{\phi}_{\nu,i,g'} + \sum_{g'=1}^{G}\Sigma_s^{i,g'\to g}\hat{\phi}_{\nu,i,g'}\right) \qquad \forall \nu \in (x,y,z) \qquad (2.57)$$

where $\hat{\phi}_{\nu,i,g}$ is the moment of the scalar flux in the ν direction. These scalar fluxes can be computed in parallel over all regions based on scalar flux moments.

2.6.1.4 Calculation of Scalar Flux Moments

The previous section allowed for the calculation of linear source moments given scalar flux moments. These scalar flux moments can be calculated by

$$\hat{\phi}_{\nu,i,g} = \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds\,\left(\nu - \nu_i^C\right)\psi_g^{t,\varsigma}(s) \qquad \forall \nu \in (x,y,z). \qquad (2.58)$$

Converting the variable ν into the tracked distance s using Eq. 2.46, this can be recast as

$$\hat{\phi}_{\nu,i,g} = \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds \left[\nu_{t,\varsigma}^m + \Omega_{\nu,t}\left(s - \frac{\ell_{t,\varsigma}}{2}\right) - \nu_i^C\right]\psi_g^{t,\varsigma}(s) \qquad \forall \nu \in (x,y,z) \quad (2.59)$$

and simplified to

$$\hat{\phi}_{\nu,i,g} = \sum_{(t,\varsigma)\in V_i} w_t \ell_{t,\varsigma}\left[\Omega_{\nu,t}\hat{\psi}_g^{t,\varsigma} + \left(\nu_{t,\varsigma}^m - \nu_i^C - \frac{\Omega_{\nu,t}\ell_{t,\varsigma}}{2}\right)\overline{\psi}_g^{t,\varsigma}\right] \qquad \forall \nu \in (x,y,z) \quad (2.60)$$

where $\overline{\psi}_g^{t,\varsigma}$ is the average angular flux which can be calculated using Eq. 2.35 and $\hat{\psi}_g^{t,\varsigma}$ is the angular flux moment defined by

$$\hat{\psi}_g^{t,\varsigma} = \int_0^{\ell_{t,\varsigma}} ds\, s \psi_g^{t,\varsigma}(s). \tag{2.61}$$

To solve for the angular flux moment, the definition for angular flux in Eq. 2.30 is inserted, yielding

$$\hat{\psi}_g^{t,\varsigma} = \int_0^{\ell_{t,\varsigma}} ds\, s \left(\psi_g^{t,\varsigma}(0) + \left(\frac{q^0_{t,\varsigma,g}}{\Sigma_t^{i,g}} - \psi_g^{t,\varsigma}(0) \right) F_1\left(\Sigma_t^{i,g} s\right) + \left(\frac{q^1_{t,\varsigma,g}}{2\left(\Sigma_t^{i,g}\right)^2} \right) F_2\left(\Sigma_t^{i,g} s, \Sigma_t^{i,g} \ell_{t,\varsigma}\right) \right) \tag{2.62}$$

which can be simplified to

$$\hat{\psi}_g^{t,\varsigma} = \frac{\psi_g^{t,\varsigma}(0)\ell_{t,\varsigma}}{2} + \left(\frac{q^0_{t,\varsigma,g}}{\Sigma_t^{i,g}} - \psi_g^{t,\varsigma}(0) \right) \frac{G_1(\Sigma_t^{i,g}\ell_{t,\varsigma})}{\Sigma_t^{i,g}} + \frac{\ell_{t,\varsigma} q^1_{t,\varsigma,g} G_2(\Sigma_t^{i,g}\ell_{t,\varsigma})}{2\left(\Sigma_t^{i,g}\right)^2} \tag{2.63}$$

where the functions $G_1(\tau)$ and $G_2(\tau)$ are mathematically defined as

$$G_1(\tau) = 1 + \frac{\tau}{2} - \left(1 + \frac{1}{\tau}\right) F_1(\tau) \tag{2.64}$$

and

$$G_2(\tau) = \frac{2}{3}\tau - \left(1 + \frac{2}{\tau}\right) G_1(\tau). \tag{2.65}$$

With these definitions, the scalar flux moments can be calculated by inserting the definitions of average angular flux $\overline{\psi}_g^{t,\varsigma}$ (Eq. 2.35) and angular flux moments $\hat{\psi}_g^{t,\varsigma}$ (Eq. 2.63) into the definition of the scalar flux moments in Eq. 2.60. This results in a long expression which, for simplicity, can be decomposed into four terms as

$$\hat{\phi}_{v,i,g} = K_1 + K_2 + K_3 + K_4 \tag{2.66}$$

where the terms are defined by:

$$K_1 = \frac{1}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i} w_t \ell_{t,\varsigma} q^0_{t,\varsigma,g} \left(\frac{\Omega_{v,t} G_1(\Sigma_t^{i,g}\ell_{t,\varsigma})}{\Sigma_t^{i,g}} + v^m_{t,\varsigma} - v_i^C - \frac{\Omega_{v,t}\ell_{t,\varsigma}}{2} \right) \tag{2.67}$$

$$K_2 = \frac{1}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i} w_t \Omega_{v,t} \ell_{t,\varsigma} \psi_g^{t,\varsigma}(0) \left(\frac{\Sigma_t^{i,g}\ell_{t,\varsigma}}{2} - G_1(\Sigma_t^{i,g}\ell_{t,\varsigma}) \right) \tag{2.68}$$

$$K_3 = \frac{1}{2\left(\Sigma_t^{i,g}\right)^2} \sum_{(t,\varsigma)\in V_i} w_t q^1_{t,\varsigma,g} \Omega_{v,t} \ell^2_{t,\varsigma} G_2(\Sigma_t^{i,g}\ell_{t,\varsigma}) \tag{2.69}$$

$$K_4 = \frac{1}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i} w_t \left(v^m_{t,\varsigma} - v_i^C - \frac{\Omega_{v,t}\ell_{t,\varsigma}}{2} \right) \left(\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma}) \right) \tag{2.70}$$

Each of the terms can be simplified. First, K_1 can be simplified by inserting the definition of $q^0_{t,\varsigma,g}$ from Eq. 2.43. After expanding terms, K_1 can be cast as

$$\begin{aligned} K_1 = & \frac{\overline{q}_{i,g}}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i} w_t \ell_{t,\varsigma} \left(v^m_{t,\varsigma} - v_i^C \right) + \frac{\overline{q}_{i,g}}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i} w_t \Omega_{v,t} \ell_{t,\varsigma} \left[\frac{G_1(\Sigma_t^{i,g}\ell_{t,\varsigma})}{\Sigma_t^{i,g}} - \frac{\ell_{t,\varsigma}}{2} \right] + \\ & \frac{1}{\Sigma_t^{i,g}} \sum_{p\in(x,y,z)} q_{p,i,g} \sum_{(t,\varsigma)\in V_i} w_t \ell_{t,\varsigma} \left(p^m_{t,\varsigma} - p_i^C \right) \left[\frac{\Omega_{v,t} G_1(\Sigma_t^{i,g}\ell_{t,\varsigma})}{\Sigma_t^{i,g}} + v^m_{t,\varsigma} - v_i^C - \frac{\Omega_{v,t}\ell_{t,\varsigma}}{2} \right] \end{aligned} \tag{2.71}$$

Note that the first term of K_1 is zero due to the definition of the centroid in Eq. 2.47. The second term also becomes zero by noting that all tracks are traversed forwards and backwards, utilizing the simplification discussed in Section 2.4. Therefore, after further expanding terms, K_1 is simplified to

$$\begin{aligned} K_1 = & \frac{1}{\Sigma_t^{i,g}} \sum_{p\in(x,y,z)} q_{p,i,g} \sum_{(t,\varsigma)\in V_i} w_t \ell_{t,\varsigma} \left(p^m_{t,\varsigma} - p_i^C \right) \left(v^m_{t,\varsigma} - v_i^C \right) + \\ & \frac{1}{\Sigma_t^{i,g}} \sum_{p\in(x,y,z)} q_{p,i,g} \sum_{(t,\varsigma)\in V_i} w_t \Omega_{v,t} \ell_{t,\varsigma} \left(p^m_{t,\varsigma} - p_i^C \right) \left[\frac{G_1(\Sigma_t^{i,g}\ell_{t,\varsigma})}{\Sigma_t^{i,g}} - \frac{\ell_{t,\varsigma}}{2} \right]. \end{aligned} \tag{2.72}$$

Notice that the bracketed quantity in the second term is not dependent on direction. Therefore, by again utilizing the forward and backward tracking relationships discussed in Section 2.4, the first term can be simplified and the second term can be eliminated, resulting in

$$K_1 = \frac{2}{\Sigma_t^{i,g}} \sum_{p\in(x,y,z)} q_{p,i,g} \sum_{(t,\varsigma)\in V_i^F} w_t \ell_{t,\varsigma} \left(p_{t,\varsigma}^m - p_i^C\right)\left(v_{t,\varsigma}^m - v_i^C\right) \tag{2.73}$$

where V_i^F represents the forward tracked volume of V_i, therefore using only half of the directional tracks. Next, K_2 can be simplified to

$$K_2 = \frac{1}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i} w_t \Omega_{v,t} \ell_{t,\varsigma} \psi_g^{t,\varsigma}(0) H(\Sigma_t^{i,g} \ell_{t,\varsigma}) \tag{2.74}$$

by defining the function H as

$$H(\tau) = \frac{\tau}{2} - G_1(\tau). \tag{2.75}$$

The K_3 term can be simplified by inserting the definition of $q_{t,\varsigma,g}^1$ in Eq. 2.44 and again using the forward and backward tracking relationships discussed in Section 2.4 as

$$K_3 = \frac{1}{\left(\Sigma_t^{i,g}\right)^2} \sum_{p\in(x,y,z)} q_{p,i,g} \sum_{(t,\varsigma)\in V_i^F} w_t \Omega_{v,t} \Omega_{p,t} \ell_{t,\varsigma}^2 G_2(\Sigma_t^{i,g} \ell_{t,\varsigma}). \tag{2.76}$$

The fourth and final term K_4 can be simplified as

$$K_4 = \frac{1}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i} w_t v_{t,\varsigma}^{\mathbf{in}} \left(\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma})\right) \tag{2.77}$$

by defining the entering coordinate $v_{t,\varsigma}^{\mathbf{in}}$ of the segment on the source region relative to the centroid as

$$v_{t,\varsigma}^{\mathbf{in}} = v_{t,\varsigma}^m - v_i^C - \frac{\Omega_{v,t}\ell_{t,\varsigma}}{2}. \tag{2.78}$$

Altogether, these equations can be combined and written compactly as

$$\begin{aligned} \hat{\phi}_{v,i,g} = {} & \frac{1}{\Sigma_t^{i,g}} \left(q_{x,i,g} C_{v,x}^{i,g} + q_{y,i,g} C_{v,y}^{i,g} + q_{z,i,g} C_{v,z}^{i,g} \right) + \\ & \frac{1}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i} w_t \left[\Omega_{v,t} \ell_{t,\varsigma} \psi_g^{t,\varsigma}(0) H(\Sigma_t^{i,g}\ell_{t,\varsigma}) + v_{t,\varsigma}^{\mathbf{in}} \left(\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma}) \right) \right] \end{aligned} \tag{2.79}$$

where for $v, p \in (x, y, z) \times (x, y, z)$

$$C_{v,p}^{i,g} = 2 \sum_{(t,\varsigma)\in V_i^F} w_t \ell_{t,\varsigma}^2 \Omega_{v,t} \Omega_{p,t} G_2(\Sigma_t^{i,g} \ell_{t,\varsigma}) + \frac{1}{\Sigma_t^{i,g}} \sum_{(t,\varsigma)\in V_i^F} w_t \ell_{t,\varsigma} \left(p_{t,\varsigma}^m - p_i^C\right)\left(v_{t,\varsigma}^m - v_i^C\right). \tag{2.80}$$

The equations derived in this section form the basis of the linear source MOC algorithm.

2.7 MOC Algorithm with Linear Sources

The linear source equations are much more complex and greater in number than the flat source equivalent, so this section will highlight the important equations and discuss how to implement an algorithm.

2.7.1 Identifying Invariant Constants

Before starting transport sweeps, it is important to identify constants which are not dependent on iterative estimates of scalar and angular fluxes. These constants can be computed and re-used at each iteration. These constants must not consume too much memory or else their storage may have significant computational drawbacks. In the context of MOC, since the number of segments is often significantly larger than the number of regions, explicit storage should be implemented for region-dependent constants and not for segment-dependent constants.

One example of region dependent constants that should be stored are the $C_{v,p}^{i,g}$ terms defined in Eq. 2.80. Note these constants are solely dependent on the geometry and track laydown which are invariant over the iterative process. These constants are therefore computed and stored for every region i.

In addition, the computation of linear source components involves solving the system described in Eq. 2.54,

$$M_i \vec{q}_{i,g} = \vec{Q}_{i,g}$$

for every region i and energy group g where $\vec{q}_{i,g} = \left[q_{x,i,g}, q_{y,i,g}, q_{z,i,g}\right]^T$ and $\vec{Q}_{i,g} = \left[Q_{x,i,g}, Q_{y,i,g}, Q_{z,i,g}\right]^T$. The matrix M_i which is a 3×3 matrix for every region i represented in Eq. 2.55 by

$$M_i = \begin{bmatrix} M_{i,xx} & M_{i,xy} & M_{i,xz} \\ M_{i,xy} & M_{i,yy} & M_{i,yz} \\ M_{i,xz} & M_{i,yz} & M_{i,zz} \end{bmatrix}$$

and the matrix components are computed with Eq. 2.53 as:

$$M_{i,v,v'} = \sum_{(t,\varsigma)\in V_i} w_t \int_0^{\ell_{t,\varsigma}} ds \left(v - v_i^C\right)\left(v' - v'^C_i\right) \qquad \forall (v,v') \in (x,y,z)\times(x,y,z)$$

Note that these terms, and therefore also M_i^{-1}, are invariant and the explicit computation and storage of M_i^{-1} is performed for every region.

The computation of M_i^{-1} should be handled with great care. For regions that are thin in a particular Cartesian direction (for instance the x direction), the associated moments in that direction can also become very small. For instance, a region thin in the x direction can have extremely small $M_{i,xx}$. This can cause the matrix to be poorly conditioned. In OpenMOC, this is handled by treating thin regions in which the single direction moments (such as $M_{i,xx}$) are small ($< 10^{-4} cm^2$) as having spatially constant source in the thin direction. In the case of a thin x direction, instead of handling the full 3×3 matrix, the system is simplified to a 2×2 matrix as

$$M_i^{-1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \multicolumn{2}{c}{\multirow{2}{*}{$\begin{bmatrix} M_{i,yy} & M_{i,yz} \\ M_{i,yz} & M_{i,zz} \end{bmatrix}^{-1}$}} \\ 0 & & \end{bmatrix}$$

which is solved to determine the y and z linear source components and the x component of the linear source is assumed to be zero. Similar simplifications are handled for thin y and thin z directions with the elimination of rows and columns involving the variable. If two directions are thin, then the source in both directions is treated as flat, with a simple 1×1 system evaluated to determine the source in the third direction. For example, for a region thin in both x and y, the system reduces to

$$M_i^{-1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/M_{i,zz} \end{bmatrix}$$

which is used to form the source in the z direction. For a region thin in all Cartesian directions, it is treated as having a flat source. In the linear source framework this means setting all elements of M_i^{-1} to zero.

In addition to region-dependent terms, the functions F_1, F_2, and H often arise in the MOC equations. These terms are dependent on optical path length and therefore are segment and group-dependent. Therefore, they are not explicitly stored since the storage requirements would be computationally infeasible. Instead, they are computed on-the-fly, often through interpolation. This will be further discussed in Chapter 3.

2.7.2 Transport Sweeps

Before each transport sweep, the source moments $Q_{v,i,g}$ can be computed from scalar flux moments using Eq. 2.52 as:

$$Q_{v,i,g} = \frac{1}{4\pi}\left(\frac{\chi_{i,g}}{k}\sum_{g'=1}^{G}\nu_{i,g'}\Sigma_f^{i,g'}\hat{\phi}_{v,i,g'} + \sum_{g'=1}^{G}\Sigma_s^{i,g'\to g}\hat{\phi}_{v,i,g'}\right) \qquad \forall v \in (x, y, z)$$

These moments are then used with Eq. 2.54 as

$$\vec{q}_{i,g} = M_i^{-1}\vec{Q}_{i,g}$$

Recall that the M_i^{-1} matrices are explicitly computed and stored before iterations. Therefore, the matrix values are simply loaded and only a matrix-vector multiplication is necessary during the computation of source components. The flat source components can then be computed in the same way as flat source MOC using Eq. 2.48 as

$$\overline{q}_{i,g} = \frac{1}{4\pi}\left(\frac{\chi_{i,g}}{k}\sum_{g'=1}^{G}\nu_{i,g'}\Sigma_f^{i,g'}\overline{\phi_{i,g'}} + \sum_{g'=1}^{G}\Sigma_s^{i,g'\to g}\overline{\phi_{i,g'}}\right).$$

The angular fluxes $\psi_g^{t,\varsigma}$ can be computed using Eq. 2.30 as

$$\psi_g^{t,\varsigma}(s) = \psi_g^{t,\varsigma}(0) + \left(\frac{q^0_{t,\varsigma,g}}{\Sigma_t^{i,g}} - \psi_g^{t,\varsigma}(0)\right)F_1\left(\Sigma_t^{i,g}s\right) + \left(\frac{q^1_{t,\varsigma,g}}{2\left(\Sigma_t^{i,g}\right)^2}\right)F_2\left(\Sigma_t^{i,g}s, \Sigma_t^{i,g}\ell_{t,\varsigma}\right)$$

where the track-based flat source $q^0_{t,\varsigma,g}$ and the track-based linear source $q^1_{t,\varsigma,g}$ are evaluated using Eq. 2.43 and Eq. 2.44:

$$q^0_{t,\varsigma,g} = \overline{q}_{i,g} + \sum_{\nu\in(x,y,z)} q_{\nu,i,g}\left(\nu^m_{t,\varsigma} - \nu^C_i\right)$$

$$q^1_{t,\varsigma,g} = \sum_{\nu\in(x,y,z)} \Omega_{\nu,t}q_{\nu,i,g}$$

The scalar fluxes $\overline{\phi_{i,g}}$ and flux moments $\hat{\phi}_{\nu,i,g}$ can therefore be calculated with Eq. 2.37 and Eq. 2.79:

$$\overline{\phi_{i,g}} = \frac{\overline{q}_{i,g}}{\Sigma_t^{i,g}} + \frac{1}{\Sigma_t^{i,g}V_i}\sum_{(t,\varsigma)\in V_i} w_t\left(\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma})\right)$$

$$\hat{\phi}_{\nu,i,g} = \frac{1}{\Sigma_t^{i,g}}\sum_{p\in(x,y,z)} q_{p,i,g}C^{i,g}_{\nu,p} + \frac{1}{\Sigma_t^{i,g}}\sum_{(t,\varsigma)\in V_i} w_t\left[\Omega_{\nu,t}\ell_{t,\varsigma}\psi_g^{t,\varsigma}(0)H(\Sigma_t^{i,g}\ell_{t,\varsigma}) + \nu^{\mathbf{in}}_{t,\varsigma}\left(\psi_g^{t,\varsigma}(0) - \psi_g^{t,\varsigma}(\ell_{t,\varsigma})\right)\right]$$

The equations presented in this section form the basis of the track-based linear source MOC implementation.

Chapter 3

Software Design and Development

The work in this thesis develops the OpenMOC [43] neutron transport code, which was developed for 2D MOC simulations, and significant work was required to extend OpenMOC to 3D MOC calculations. This chapter explains the structure of OpenMOC and the changes that were necessary to increase code flexibility and accommodate 3D MOC calculations.

3.1 OpenMOC Overview

OpenMOC is neutron transport code that is written in C++ with a Simplified Wrapper Interface Generator (SWIG) [44] to expose the C++ classes and routines to the Python scripting language. This allows users to take advantage of the simplicity and flexibility of the Python language while also having the performance benefits of C++ compiled code. In this way, users can work entirely in Python without having to touch the underlying C++ code. In addition, users do not have to learn a new input file syntax, only the names, functionality, and input variables of functions constituting the Application Programming Interface (API). This allows for users to write code with greater ease.

The underlying C++ code of OpenMOC also leverages the use of OpenMP [45] for shared memory parallelism. In this framework, all on-node data is shared between threads. With the emergence of the 3D solver, distributed parallelism has also been implemented with MPI [14] in the form of domain decomposition, discussed in Chapter 6.

With this hybrid parallelism design, OpenMOC is able to scale to both many CPU cores and many nodes.

OpenMOC is built on the use of Constructive Solid Geometry (CSG), which allows complex geometries to be built out of boolean operations – such as intersections and unions – of simple surfaces and building blocks termed *primitives*. In addition, a hierarchy is used to agglomerate collections of primitives together. This approach is particularly useful for reactor geometries which are often highly structured. For example, a typical reactor core is built out of simple *fuel pins*, each describing the geometric detail of a fuel rod, which are grouped together into *assemblies*. Assemblies are then grouped together to form the reactor core. An example of a 2D CSG construction of a single assembly is given in Figure 3-1.

One of the benefits of the CSG approach is a reduced memory requirements of storing the geometry. Instead of explicitly storing the geometry information of each fuel pin within the reactor core, only unique fuel pin types need to be stored. They are then referenced by their parent structure. For instance, an assembly containing a lattice of fuel pins creates a mapping of lattice location to the unique fuel pin type, rather than replicating the full geometric information for each fuel pin.

In addition, the formation of a CSG allows ray tracing to be conducted in a general framework, agnostic of the individual primitives. Each *cell* in OpenMOC is defined in terms of *surface* objects and the half-space of each surface. A half-space determines on which side of the surface the cell is located. Ray tracing fundamentally involves calculating the distance to intersection along a direction. With the CSG framework, each of the bounding surfaces is queried for the distance to intersection. This naturally speeds up ray tracing by not having to check each instance of a surface within the geometry, but rather only the local surfaces.

Once the geometry is built, OpenMOC generates tracks across the constructed geometry, and solves the neutron transport equation iteratively. The solver can then be queried to return the scalar flux distribution as well as reaction rates. In order to visualize the data, OpenMOC includes Python plotting routines for scalar flux data, computed reaction rates, as well as geometric detail and visual diagnostics.

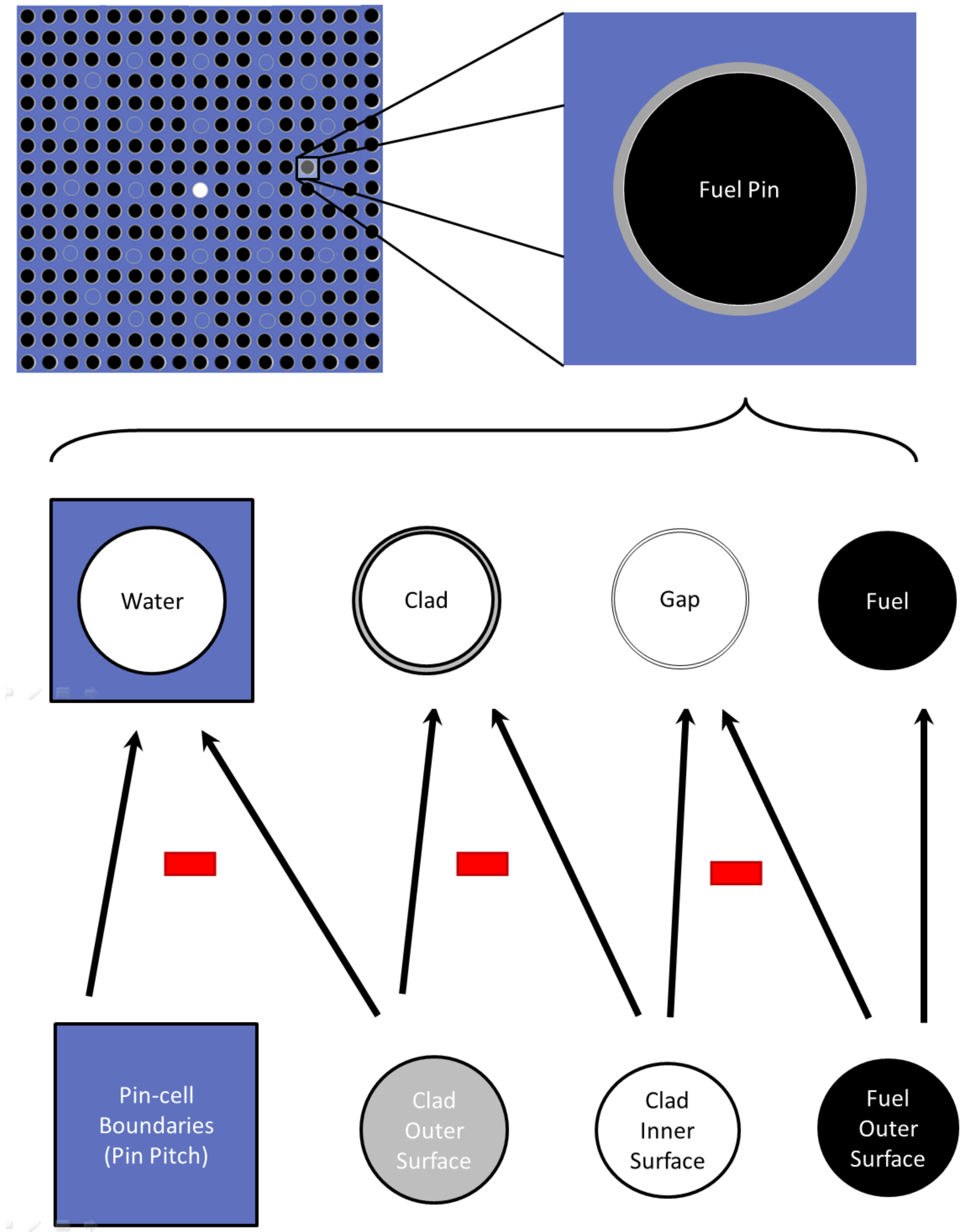


Figure 3-1: The hierarchical CSG construction of a typical assembly.

The interested reader can find a more complete description of the OpenMOC code in Boyd's thesis [46] in which the object-oriented structure and geometry handling is explicitly defined.

3.2 Object Oriented Design

OpenMOC uses the object oriented programming paradigm whereby data structures called *classes* are created that encapsulate both the data and associated subroutines. Object oriented programming generally leads to more resilient code since only the class itself can access its private attributes. An instantiation of a class is termed an *object*. OpenMOC applies many of the principles of object oriented programming including information hiding, inheritance, and polymorphism.

An OpenMOC simulation requires three main components: a geometry, a track generator, and a solver. All of these are C++ classes in OpenMOC are exposed to the user. The user first describes the surfaces, cells, universes, and materials which constitute the geometry in a hierarchical CSG arrangement. The user then instantiates a `Geometry` object and provides it the root cell of the CSG. Next, a `TrackGenerator` is instantiated which is provided the `Geometry` object as well as track generation parameters such as radial ray spacing and number of azimuthal angles. Lastly, a `Solver` object is instantiated and provided the `TrackGenerator` object along with convergence criteria. The solver can then be called to solve the MOC equations, such as the MOC neutron transport eigenvalue problem, using the track and segmentation information found in the `TrackGenerator`.

Extending the OpenMOC solver to 3D simulations required restructuring all of these classes in order to make them more flexible. A focus in extending the sovler to 3D simulations was to still maintain the ability to run 2D simulations. A benefit of maintaining the ability to run 2D simulations is to allow the user to compare 2D and 3D simulations on the same problem. To accommodate this, the input structure between 2D and 3D simulations is very similar. In order to make the code more resilient and simpler, common code reuse was emphasized. Many of the routines present in the 2D simulations are also used in the 3D simulations so both should use the same code. These points illustrate the need for maximum cohesion between the 2D and 3D solver modes. This was accomplished by expanding the 2D classes to be more general.

3.2.1 `Geometry` Class Updates

The `Geometry` class was altered to accommodate piecewise *axially extruded geometries*. Axially extruded geometries are geometric configurations in which the geometry looks the same at every axial level. A piecewise axially extruded geometry is a geometry that can be formed as the union of a finite number of extruded geometries. For instance, a fuel rod with end caps would fit the description of a piecewise axially extruded geometry but a sphere would not. Most practical reactor applications are indeed piecewise extruded geometries so this is not a very strong limitation.

With the change from 2D geometries to piecewise axially extruded geometries, circles are transformed to z-cylinders (cylinders with a vertical major axis) and z-planes are added with a similar structure to the x and y planes already incorporated in OpenMOC. With this new geometry paradigm, 2D problems are thought of as simulating a radial slice of a 3D geometry at a given z height.

3.2.2 `TrackGenerator` Class Updates

The `TrackGenerator` class was updated to generate tracks and ray trace, compatible with the updated `Geometry` class. 2D ray tracing is imagined as ray tracing tracing perpendicular to the z-axis at some z-height. By default this height is assumed to be 0.0 in order to limit the complexity of user input for 2D simulations. The user can specify a different z-height using the `setZCoord` function of the `TrackGenerator` class.

For 3D simulations, a new `TrackGenerator3D` class was created which inherits from the `TrackGenerator` class. In object oriented programming, a class that inherits from a parent class has access to all the parent class data and subroutines. Since tracks are built on a 2D projection, the 3D track generator must have all the functionality of the regular 2D track generator. This is the typical paradigm where class inheritance is useful.

During ray tracing, a `TrackGenerator3D` object first ray traces all 2D tracks (determined from the parent `TrackGenerator` functionality) over all potentially unique z-planes in the geometry to form the equivalent of a ray trace over the composite of all radial detail in the geometry. Since this can potentially be expensive, users are allowed to indicate the unique piecewise axially extruded ranges in the `Geometry` through the `setSegmentationZones` function in the `TrackGenerator3D` class. If these zones are not specified, all z-planes in the geometry are viewed as a potential divider between different axially extruded regions, leading to longer setup time.

Once the 2D ray trace is conducted over all axially extruded regions, the full 3D ray trace can be calculated. Due to the expense of explicitly storing all 3D tracks in a typical geometry, the `TrackGenerator3D` class generates tracks on-the-fly. Similarly, segments can be prohibitively expensive to store. Therefore, the default mode is to calculate segments on-the-fly rather than upfront, although both options are available. Ray tracing and segmentation is discussed in greater detail in Chapter 5.

3.2.3 `Solver` Class Updates

The `Solver` class has also been reformulated to support both 2D and 3D simulations. The `Solver` class is an abstract class, meaning it contains the description of data and associated subroutines, but cannot be instantiated on its own. Instead, there are subclasses that inherit from the abstract class that can be instantiated. In this case the `CPUSolver` and `GPUSolver` classes are both subclasses of the `Solver` class. Rather than supporting both the `CPUSolver` and `GPUSolver` classes, this thesis focuses on the `CPUSolver` class to allow for easier implementation of the object oriented structures.

The `CPUSolver` was altered to support the calculation of both 2D MOC and 3D MOC equations. By referring to the supplied `TrackGenerator` object, it can determine whether a 2D or 3D simulation should be calculated. If a `TrackGenerator3D` object was provided to the solver, it runs a 3D simulation. Otherwise, a 2D simulation is run. While much code is shared between the 2D and 3D simulations in the solver, the code which calculates the variation of angular flux over segments has separate 2D MOC and 3D MOC code sections in order to maximize performance.

In addition to the existing `CPUSolver`, a new `CPULSSolver` class has been added to OpenMOC which is capable of using a linear source approximation. Since the linear source solver needs much of the code present in the regular flat source solver, `CPULSSolver` was implemented as a subclass of `CPUSolver`.

3.3 Modular Structure

In altering OpenMOC to support both 2D and 3D simulations, it was quickly realized that without a massive overhaul, there would be a very large amount of repeated code. The goal was to create code that was capable of conducting both 2D and 3D MOC simulations, but also to compare multiple types of ray tracing and mathematical approximations (such as linear source). For all these options to be supported, there is the possibility for much of the same functionality to be implemented at multiple points in the code.

For example, MOC can largely be described as an algorithm that performs a ray trace and then computes equations over the segments formed from the ray trace. Many different ray tracing algorithms could be used with the underlying equations and solver remaining theoretically unchanged. However, if the code is rigid, each new ray tracing algorithm would require an entire code re-write.

Therefore, a goal in developing OpenMOC to support work presented in this thesis is to maximize code reuse. A structure was created in which segment traversal, ray tracing, and the algorithms that use segment information could be easily decoupled. In this new structure, `MOCKernel` classes dictate what operations to perform on each segment and a `TraverseSegments` class dictates how to iterate over segments and which ray tracing method to use. Both of these classes are virtual classes which have subclass implementations. Specific algorithms that use segment information are formed as subclasses of the `TraverseSegments` class, each defining operations to perform on groups of segments. The algorithms should also specify which `MOCKernel` objects to use.

3.3.1 `MOCKernel` Classes

Since the `MOCKernel` class operates directly on segments, it appears in the inner-most loop in algorithms. The `MOCKernel` parent class contains common information used for most of its subclass implementations. These attributes include:

- A counter for the number of segments encountered by the kernel
- The maximum path length allowed for segments before they must be split
- The number of energy groups

An `MOCKernel` class also has a few virtual functions that must be implemented by subclasses. Notable functions include:

- An `execute` function which dictates what to do on a segment
- A `newTrack` function that dictates functionality when a new group of segments is encountered.

The `execute` function is applied immediately when a segment is formed from ray tracing (or loaded in the case of explicitly stored segments) whereas the `newTrack` function is applied at the start of sequencing through a group of segments. The segments comprising a track is an example of a common grouping of segments, though other groupings could be chosen.

Since the `MOCKernel` class is a virtual class, it is meant to have subclasses which can be instantiated. The current `MOCKernel` subclasses implemented in OpenMOC are:

- `CounterKernel`: Counts the number of segments encountered by the kernel
- `VolumeKernel`: Uses track weights and segment lengths to form an estimate of the volumes of regions encountered
- `TransportKernel`: Directly applies the transport equation on the encountered segments
- `SegmentationKernel`: Caches segment information for use in later calculations

Since the `SegmentationKernel` object just caches information it is useful for complicated operations or operations which can be most efficient when applied to a bulk grouping of data. For this efficiency reason, the `TransportKernel` is not currently used in the default solver. Instead, a `SegmentationKernel` is used in its place so that the operations can be applied to flux data at once instead of interleaving ray tracing with transport calculations.

3.3.2 `TraverseSegments` Classes

Unlike the `MOCKernel` parent class, the `TraverseSegments` virtual class contains a significant amount of algorithmic information. Specifically, the virtual class contains all segment iteration and ray tracing algorithms. The algorithm has a `loopOverTracks` function that takes an `MOCKernel` as an argument. Subclasses of `TraverseSegments` can call this function which iterates over groups of segments (eg. tracks).

Inside the `loopOverTracks` function, a conditional block calls the appropriate segment looping scheme, passing along the provided kernel. The currently implemented segment iteration schemes are:

- `loopOverTracks2D`: Iteration scheme for looping over explicitly stored 2D segments
- `loopOverTracks3DExplicit`: Iteration scheme for looping over explicitly stored 3D segments
- `loopOverTracksByTrackOTF`: Iteration scheme for looping over and generating segments on-the-fly where segments with ray tracin by individual track, described in Chapter 5
- `loopOverTracksByStackOTF`: Iteration scheme for looping over and generating segments on-the-fly with ray tracing by z-stack, described in Chapter 5

Each of the iteration schemes loops over all tracks and segments in the problem in some order, performing the associated ray tracing algorithm when appropriate. The ray tracing scheme is provided information relevant to ray tracing or loading the segments of interest. Before calling the ray tracing scheme, the `newTrack` function of the provided kernel is called. Then the ray tracing scheme is called and provided the kernel so that it can call the `execute` function immediately when the segment is formed or loaded.

After the ray tracing scheme is completed, an `onTrack` function is called which details operations to perform on the group of segments. The cached segment information is also provided if a `SegmentationKernel` was used. The transport sweep algorithm currently used in OpenMOC applies all of the MOC equations inside the `onTrack` function using the provided cached segment information.

In general, algorithms which require track or segment information should form a small subclass of the `TraverseSegments` function and define the `onTrack` function. Additionally, an `execute` function should be defined which details the general structure of the algorithm. It should include the setup, call the `loopOverTracks` function to loop over all segments and apply the algorithm defined by the `onTrack` function and the provided kernel, and then detail any operations to perform after looping over all tracks and segments.

An example of an execute function is given below for the `VolumeCalculator` class which calculates the volumes of all regions in the geometry. Since it performs all operations directly on segments using `VolumeKernel` objects, the `onTrack` function is defined to simply return.

```
void VolumeCalculator::execute() {
        #pragma omp parallel
        {
                MOCKernel* kernel = getKernel<VolumeKernel>();
                loopOverTracks(kernel);
        }
}
```

For algorithms in which only operations need to be performed on tracks, a `NULL` kernel can be provided which alerts the segment iteration scheme to skip the ray tracing step. Numerous subclasses of `TraverseSegments` are present in OpenMOC since every algorithm which loops over segments or tracks is contained in a subclass of `TraverseSegments`. They all appear in the files `TrackTraversingAlgorithms.h` and `TrackTraversingAlgorithms.cpp`.

3.4 Computing Systems

In this thesis, all computational results either use the Argonne BlueGene/Q supercomputer or the Falcon supercomputer at the Idaho National Laboratory. An overview of system parameters is shown in Table 3.1. Note that the BlueGene/Q architecture have significantly lower clock speeds. In addition, these cores have limited branch prediction capabilities, causing them to be much slower. The Argonne BlueGene/Q supercomputer contains two partitions of interest in this thesis: the Cetus partition and the Mira partition. The Cetus partition is intended for small simulations whereas the Mira partition is intended for large simulations.

Table 3.1: Description of single node supercomputer architectures

	Argonne BlueGene/Q	Falcon
CPU Type	Single Socket	Dual Socket
CPU Name	BlueGene/Q	E5-2695 v4
Architecture	PowerPC A2	Broadwell
Cores per Node	16	36
Hardware Threads per Node	64	36
Speed (GHz)	1.6	2.1
Node Memory (GB)	16	128

3.5 Performance Considerations

So far this chapter has emphasized software design for code resiliency and flexibility. However, a useful application should also minimize runtime. This section concentrates on software design considerations to maximize performance and minimize runtime.

3.5.1 Addressing Performance in Object-Oriented Modular Software Design

While the object-oriented and modular design is useful for code reuse, minimizing the number of potential bugs and allowing for greater flexibility, it does add some computational overhead. The overhead is largely due to added function calls. This can be overcome through the use of *inline functions* which inject their code directly into sections where they are called at compile time. While this increases the performance of the code, it can radically increase the size of the executable, especially for large inline functions. Therefore, functions should only be inline functions if they are small.

However, successfully using inline functions can sometimes be difficult where logic is required to determine which function to call. Therefore, performance-sensitive code should be designed such that the number of function calls which are not inline functions are minimized.

The optimal code design blends object oriented and modular design for code that is not performance-critical while using optimized and less flexible code for the performance-sensitive sections. In terms of the MOC algorithm, over 95% of the runtime is typically spent applying the MOC equations for segments. In OpenMOC, the application of the MOC equations occurs in the `CPUSolver::tallyScalarFlux` function. This function, which is quite small in size, should be thoroughly optimized for performance while the rest of the code should rely heavily on object oriented and modular design principles.

3.5.2 Data Organization

One critical consideration when designing performance-sensitive code is the memory layout. When CPU cores load information from memory, they load entire cache lines into local caches rather than just the desired variable. This allows the number of loads to be reduced if nearby memory is accessed. When desired memory is already within cache, it is termed a *cache hit*. If it is not, it is termed a *cache miss*, requiring an extra load. Optimizing cache performance to maximize cache hits is critical when designing efficient software, as loads from main memory can be quite expensive. *Locality* refers to how well memory is organized such that nearby or local memory is accessed frequently.

To maximize locality in MOC applications, most data should be structured to be contiguous in energy groups. Generally, the MOC equations can be approached as being applied to segments over energy groups. Since MOC tracks traverse regions in different orderings, it is difficult to optimize the cache efficiency of data accessed by region. However, the data for each region can easily be structured and always traversed in the same ordering. Therefore, the maximal amount of work should be completed on each region before moving to the next. This implies that the energy groups should all be treated together and the data structures should be contiguous in energy groups to reflect this memory access pattern.

For linear source MOC, the data is laid out in a slightly different way. While it is still important for energy groups to be close in memory for linear source solvers, multiple quantities are tallied and used for each energy group due to the addition of moments in each Cartesian direction. Therefore, scalar flux moments and source moments are laid out contiguous in Cartesian direction (x, y, and z). Specifically, the arrays of scalar flux moments and source moments are indexed by $i \times 3G + 3 \times g + d$ where i is the region ID, g is the energy group, and d represents the x, y, and z directions.

3.5.3 Scratch Pads for Temporary Storage

There are several points throughout the MOC algorithm where quantities need to be temporarily stored by region or by group. One example is calculating total reaction rates. To compute a total reaction rate in the reactor, the reaction rates from all regions and energy groups need to be calculated and summed. It is possible to add each contribution from a region and group to a tally for the total reaction rate immediately after calculation, but this can lead to significant roundoff error. Alternatively, if the summed values could be first stored in an array and then added in a pairwise fashion, the roundoff error can be significantly reduced [47].

The number of summed quantities is NG where N is the number of source regions and G is the number of energy groups. An array of NG could be allocated, but this would require a significant amount of additional storage. Instead, two arrays are allocated – one of size G and another of size N. Reaction rates for all G groups for an individual region are computed and then stored in a temporary array of size G. This array is then summed using a pairwise sum. The result is then stored in an array of size N. After this conducted for all N regions, the array of size N is then summed in a pairwise fashion.

Since temporary arrays of size N and size G are often found throughout OpenMOC, memory for these arrays is allocated at the beginning of the solver for each thread. Allocating memory can often be quite costly. If the arrays were allocated just before use, there would be a significant number of memory allocations. By allocating the memory upfront and later referencing the allocated memory, the number of memory allocations is minimized, improving performance.

3.5.4 Minimizing Parallel Contention

When multiple threads modify shared data, bugs can be incurred due to race conditions where the outcome of a program is dependent on the order in which threads execute operations. For instance, consider multiple threads adding numbers to a shared counter. The fundamental operations that access the data are reads and writes. Therefore, one thread could read a value of the counter and before the thread adds the read value with its added value, the counter updates. The thread then writes the computed value to the shared counter, disregarding the update that occurred after the read, leading to an error in the computation. This occurs because the read, addition, and write are separate operations where the data can be modified between the operations [48].

Because of these considerations, care must be taken to ensure correctness in multi-threaded applications. A common way to ensure correctness is to place a *lock* around *critical sections*. A critical section is where shared data is potentially modified by multiple threads. A lock ensures that at most one thread can access the critical section at any time. While locks do ensure correctness, they can also cause a bottleneck in the code, especially when many threads are present and attempting to access the same critical section. This situation is termed *contention*.

In terms of the MOC algorithm, in which each thread is responsible for applying the MOC equations to a group of segments, threads can be handling segments which access and modify data (such as the scalar flux) for the same region. A naive implementation would place a lock around the modification of the local scalar flux data for each energy group when the segment's contribution is computed. While this ensures correctness, it requires that each thread attempt to acquire a lock for each energy group, which can incur significant overhead cost.

Instead, each thread tallies the contributions to each energy group in its local temporary storage array, described in the previous section. This allows for the thread to modify the array without any contention with other threads. After the contributions are computed and stored for every energy group, a lock is acquired for the region and the local thread contributions are summed to the global tally of scalar flux. This allows for only one lock to be acquired for each segment traversal. In addition, it allows for increased cache efficiency and vectorization. Lock operations require communication between cache levels to ensure correctness and are not vectorizable. By decreasing the frequency to which locks are accessed, these issues can be minimized. This was thoroughly tested during the development of a prototype application named SimpleMOC [49].

3.5.5 Computing Exponentials

One major bottleneck of MOC solvers, particularly for flat source MOC, is the computation of exponentials. Since an exponential term is present in the MOC equations, dependent on optical path length, an exponential needs to be computed for each segment and group. Specifically, $1 - e^{-\tau}$ needs to be computed, where τ is the optical path length.

A naive approach would just use the intrinsic `exp` function available in the standard C++ library. While this would accurately compute the exponential, the intrinsic `exp` function can be quite expensive. Boyd showed that for the initial flat source implementation of OpenMOC, it was far more efficient to compute exponentials with a table interpolation [46].

OpenMOC calculated exponentials with a table interpolation instead of using the intrinsic exponential function. One important consideration with MOC implementations is the behavior in optically thin regions. For instance, consider the flat source equation

$$\psi_g^{t,\varsigma}(\ell_{t,\varsigma}) = \psi_g^{t,\varsigma}(0) + \left(\frac{q_{i,g}^0}{\Sigma_t^{i,g}} - \psi_g^{t,\varsigma}(0) \right) F_1\left(\Sigma_t^{i,g} \ell_{t,\varsigma}\right)$$

in which the total cross-section $\Sigma_t^{i,g}$ becomes zero. The fundamental MOC equations still apply, but this form poses computational difficulties as the total cross-section is in the denominator. However, this equation can be recast as

$$\psi_g^{t,\varsigma}(s) = \psi_g^{t,\varsigma}(0) + \left(q_{i,g}^0 \ell_{t,\varsigma} - \left[\Sigma_t^{i,g}\ell_{t,\varsigma}\right]\psi_g^{t,\varsigma}(0)\right)\frac{F_1\left(\Sigma_t^{i,g}\ell_{t,\varsigma}\right)}{\Sigma_t^{i,g}\ell_{t,\varsigma}} \tag{3.1}$$

and instead of building a table of F_1 which is interpolated, the table is built for the term $F_1(\tau)/\tau$ where τ is the optical path length $\Sigma_t^{i,g}\ell_{t,\varsigma}$. When calculating this term for low optical path length during table formation, a quadratic expansion around zero is used rather than the direct computation of $F_1(\tau)/\tau$. Note that the limit of $F_1(\tau)/\tau$ at zero is 1.0. Similarly, the exponential terms that arise in linear source (such as F_2) are also computed in a similar fashion ($F_2(\tau)/\tau$) with a quadratic expansion for low optical path length.

While the interpolation approach has been used by many, others have implemented specialized exponential functions which can more efficiently calculate exponentials [50]. These specialized exponential functions allow for variable precision, which is useful for the MOC application where reduced precision on exponential terms can still maintain solution accuracy.

In comparison with table interpolation, the specialized exponential functions trade a decrease in loads for an increase in Floating Point Operations (FLOPs). In recent years, the computational capabilities of computing FLOPs have increased while the performance of loading from memory has plateaued [51]. Therefore, this trade-off can be beneficial.

For flat source MOC implementations, the specialized exponential functions can indeed lead to computational improvements. However, for linear source MOC implementations, the exponential terms F_2 and H need to be computed in addition to F_1. These two additional terms have far more complicated analytic forms than F_1. Therefore, the increased FLOP work induced from computing exponentials explicitly is far greater for linear source than flat source MOC. For this reason, table interpolation is used for all three exponential terms (F_1, F_2, and H) in the linear source implementation. They are stored sequentially in memory for each interpolation point such that all three exponential terms can be retrieved with minimal loads. Again, they are stored in terms of $F_1(\tau)/\tau$, $F_2(\tau)/\tau$, and $H(\tau)/\tau$ for numerical stability using a quadratic expansion for low optical path length.

Both the specialized exponentials and table interpolation were implemented for flat source MOC whereas only table interpolation was implemented for linear source MOC. To test the impact of the different formulations on the computational profile of the solver, the SDSA problem detailed in Appendix E.2.5 is simulated using the parameters presented in Table 3.2.

Table 3.2: MOC ray parameters for the SDSA test problem for computational profiling

Radial Ray Spacing	0.1 cm
Axial Ray Spacing	1.5 cm
Number of Azimuthal Angles	16
Number of Polar Angles	4

These parameters are selected to be quite coarse in order to feasibly run in-depth profiling diagnostics. In particular, the problem is simulated in Intel VTune to record the number of floating point operations performed. The floating point utilization, often represented in GFLOPs per second, shows how well the implementation is using the available floating point units. Since usable code cannot achieve optimal floating point performance due to the need to load and store information, the LINPACK benchmark is often used for comparison. The LINPACK benchmark represents dense linear algebra computation, in which floating point units are well suited. Many regard LINPACK as the maximum floating point utilization achievable in practice. The results are shown in Table 3.3 with only one test conducted per configuration, yielding no error estimates. Note that all cases use on-the-fly ray tracing by z-stack, explained later in Chapter 5. In this thesis, the integration time is often used as a performance metric, referring to the time required to compute the angular flux variation over a segment and tally its contribution to the local scalar flux for a single energy group. It is calculated as

$$\frac{TN_C}{N_I}$$

where T is the average time to compute one transport sweep, N_C is the number of CPU cores used, and N_I is the number of integrations, which can be computed as $N_I = 2SG$ where S is the number of segments and G is the number of energy groups. The factor of two arises since each segment represents forward and backward directed angular fluxes. All results also use one node of the Falcon supercomputer, utilizing all 36 cores with shared memory parallelism.

Table 3.3: Computational profiles of flat and linear source solvers on a single node of the Falcon supercomputer

Source Approximation	Exponential Computation	Integration Time (ns)	GFLOPs/s	% LINPACK
Flat	Sp. Function	8.4	56.49	5.4
Flat	Interpolation	9.7	23.56	2.3
Linear	Interpolation	27.1	36.7	3.5

These results show that linear source incurs a $\approx 2-3\times$ performance overhead. Since the specialized exponential function incurs added FLOP work, it allows for greater floating point utilization. For the exponential interpolation implementations, the floating point utilization is significantly lower and performance is instead heavily reliant on the speed of loading from nearby caches. The linear source solver has greater floating point utilization than the associated flat source solver due to the equations being significantly more complicated, leading to increased FLOP work. Note that both flat source implementations are able to achieve less than 10 ns integration times on the Falcon supercomputer. In order to simplify the code, the specialized exponentials were later removed from the code base, as they are not useful in the targeted linear source solver.

3.5.6 Floating Point Precision

Another performance consideration is the precision to use for floating point variables – either single or double precision. In general, all accumulators, which store the summation of many terms, are stored in double precision because they are sensitive to roundoff error. Common examples include the eigenvalue k and residual estimates. On the other hand, values which are less likely to have an impact on results due to roundoff error are stored in single precision. For instance, the angular fluxes are always stored in single precision. This allows for the memory footprint to be reduced and cache efficiency to be maximized as twice as much data can be stored in cache.

Some variables allow for either single or double precision, depending on specified compiler options. Specifically, the types `FP_PRECISION`, `SF_PRECISION`, and `CMFD_PRECISION` are all defined to be either single or double precision floating point types at compile time. The `FP_PRECISION` type dictates the precision of the most accessed data, including the exponential table and cross-section data. The `SF_PRECISION` type dictates the precision of scalar flux data and the `CMFD_PRECISION` dictates the precision of CMFD matrices and linear algebra solvers.

The `SF_PRECISION` type, dictating the precision of scalar flux data is defined separate from `FP_PRECISION` to independently treat the way scalar flux tallies are handled. On one hand, scalar flux data can be viewed as accumulators since many segments tally their contribution to the scalar flux in each region. However, the data footprint of scalar flux data can also be significant and impact the performance of the solver.

For high performance computing problems, all the precision types are typically set to single precision in order to maximize performance. However, when very tight convergence is desired, for debugging or convergence analysis, all precision types are set to double precision. In this thesis, all timing results presented use single precision, but double precision is used for cases where convergence behavior is analyzed to tight convergence.

To test the effect of floating point precision, simulations of the SDSA test problem described in Appendix E.2.5 are conducted with the modular track laydown which will be introduced in Chapter 4, on-the-fly ray tracing by z-stack introduced in Chapter 5, and with the linear source approximation described in Chapter 2. The aim is to show performance differences from using single or double precision. Here, only one transport sweep iteration is conducted in each test. The MOC parameters used in the tests are those presented later in Table 8.7 and the results are presented in Table 3.4. In these tests, the reported precision refers to the precision of both `SF_PRECISION` and `FP_PRECISION`. The tests were conducted using all 36 cores on one node of the Falcon supercomputer. For each precision, three simulations were conducted with the runtime uncertainty reported as the maximum difference from the mean of the three tests.

Table 3.4: Effect of floating point precision on performance of the SDSA test problem

	Single Precision	Double Precision
Total Memory (GB)	12.49	12.81
Boundary Angular Flux Storage (GB)	12.15	12.15
Scalar Flux Storage (GB)	0.39	0.76
Scalar Flux Moments Storage (GB)	0.78	1.52
Runtime (sec)	98 +/- 7	100 +/- 6

Note that boundary angular fluxes are always stored in single precision. The single precision boundary angular flux storage accounts for 94.8% of total memory when the `SF_PRECISION` and `FP_PRECISION` types set to double precision. When these types are set to single precision, boundary angular flux storage accounts for 97.3% of total memory. Since boundary angular flux storage accounts for the vast majority of memory consumption, the memory footprint of OpenMOC would almost directly double if the boundary angular fluxes were stored in double precision.

The effect of single or double precision for the `SF_PRECISION` and `FP_PRECISION` types on runtime is not obvious, as the difference is less than the reported uncertainties. The total memory footprint is only slightly reduced by using single precision. The effect of CMFD precision on runtime is not tested in these studies, but the final results presented in this thesis will show the computational cost of the CMFD solver to be insignificant for the targeted full core simulations with MOC parameters chosen to sufficiently converge the solution is space and angle.

3.5.7 Organizing Looping Structures

With the linear source approximation, the amount of computational work is significantly increased per segment traversal. The algorithm is split into tight loops over specific operations that access similar memory. This was shown to have performance benefits in the early prototype work [49]. Specifically, the algorithm presented in Alg. 3-1 is used for each segment traversal during transport sweeps. Note that the temporary buffers B_k for $1 \leq k \leq 7$ are allocated at the beginning of the simulation for each thread in order to reduce the number of allocations and reduce thread contention.

Algorithm 3-1: Kernel for applying MOC linear source equations on a segment

```
Consider a segment $\varsigma$ on track $t$ traversing region $i$ with length $\ell_{t,\varsigma}$
$G$ is the number of energy groups
$\Sigma_t^{i,g}$ represents the total cross-section for each group $g$
$\overline{\phi_{i,g}}$ is the array of average scalar fluxes
$\hat{\phi}_{v,i,g}$ is the array of scalar flux moments for each Cartesian direction $v$
$\psi_g^t$ is the array of current angular fluxes along the track
$B_1$, $B_2$, $B_3$, $B_4$, $B_5$, and $B_6$ are arrays of length $G$
$B_7$ is an array of length $3G$
for all $g \in G$ do                              ▷ Loop over all groups and compute exponentials
    $\tau \leftarrow \Sigma_t^{i,g} \ell_{t,\varsigma}$
    $B_1[g] \leftarrow F_1(\tau)$                                                        ▷ Eq. 2.25
    $B_2[g] \leftarrow F_2(\tau)$                                                        ▷ Eq. 2.31
    $B_3[g] \leftarrow H(\tau)$                                                          ▷ Eq. 2.75
end for
for all $g \in G$ do          ▷ Loop over all groups and compute track-based source terms
    Compute track-based flat source $q^0_{t,\varsigma,g}$                                ▷ Eq. 2.43
    Compute track-based linear source $q^1_{t,\varsigma,g}$                              ▷ Eq. 2.44
    $B_4[g] \leftarrow q^0_{t,\varsigma,g}$
    $B_5[g] \leftarrow q^1_{t,\varsigma,g}$
end for
Compute direction independent terms in Eq. 2.31
for all $g \in G$ do        ▷ Loop over all groups and apply linear source MOC equations
    Using arrays $B_1$, $B_2$, $B_3$, $B_4$, and $B_5$ compute the change in angular flux
    Tally contribution to local average scalar flux tally $B_6[g]$                       ▷ Eq. 2.81
    for all $v \in (x, y, z)$ do
        Directional index $d$ maps $v \in (x, y, z)$ to $d \in (0, 1, 2)$
        Calculate contribution to local scalar flux moment tally                       ▷ Eq. 2.81
        Tally contribution to array $B_7$ at $B_7[3g + d]$
    end for
    Update angular flux $\psi_g^t$
end for
Acquire lock for region $i$
for all $g \in G$ do                  ▷ Loop over all groups and tally contributions to fluxes
    Tally average scalar flux contributions in $B_6$ to $\overline{\phi_{i,g}}$
    Tally average scalar flux contributions in $B_7$ to $\hat{\phi}_{v,i,g}$
end for
Release lock for region $i$
```

3.6 User Input

In terms of input structure, the 3D MOC updates to OpenMOC were structured to minimize the amount of work required to convert a 2D input to a 3D input. From a fully defined 3D geometry, including z-planes, the only difference in input between a 2D and 3D simulation is the definition of the track generator. A regular 2D `TrackGenerator` object supplied to a solver will trigger the 2D MOC solver, whereas a `TrackGenerator3D` object will trigger the 3D MOC solver. Similarly, a `CPUSolver` object triggers the flat source solver, whereas a `CPULSSolver` object triggers the linear source solver.

If the Geometry was written in two dimensions without specifying z-planes, the geometry will need to be bounded in order to have a well defined 3D MOC problem. In theory, a 2D simulation implies infinite dimensions axially so if a bounded 3D geometry is provided to a regular 2D `TrackGenerator` object, a warning will be triggered, but the simulation will still run assuming infinite axial dimensions.

To facilitate running on systems which do not have thorough Python support or have difficulty transferring MPI communicator objects between Python and C++ for domain decomposed simulations, an alternative C++ build was implemented in the `profile` directory. This build uses a standard C++ Makefile and sample C++ inputs are provided as well. This alternative build might also be advantageous for developers that would like quick turnaround between updates in the core OpenMOC code rather than having to re-install OpenMOC after each source code update.

The downside of using the C++ build is that a C++ input file is required, which can be quite inflexible compared with regular scripting languages such as Python. This difficulty mainly arises during geometry and material definitions. Therefore, new routines were created in the `Geometry` class that save the geometry and material descriptions to a geometry binary file (usually with the *.geo extension). The file can then be easily loaded in either Python or C++ using the OpenMOC geometry reader. This allows the user to take advantage of both the flexibility of the Python build for inputs and the C++ build for universality and ease of installation. This process of creating geometry and material definitions in Python while running the simulations using the C++ build was used for the vast majority of the results presented in this Thesis.

3.7 Version Control and Licensing

OpenMOC utilizes Git version control and an open source distribution is hosted on GitHub at `https://github.com/mit-crpg/OpenMOC.git`. Git is a free and open source version control distribution that is becoming the software industry standard for version control. GitHub uses the Git distribution to host software distributions, both open source and closed source. Pull requests to the *develop* branch form the basis by which the code evolves. Anyone in the public can contribute to the code by making a pull request to the develop branch. The work presented in this thesis is found on the *3DMOC* branch, which has not yet been merged with the *develop* branch since it does not contain all the functionality currently available on the *develop* branch for 2D MOC simulations.

The OpenMOC code has been approved for open source release by the MIT Technology Licensing Office (TLO) under the MIT/X license. This license allows anyone to download the software without restriction. In addition, modifications to the software may be published, distributed, or sold. OpenMOC is designed with the intent of experimenting with new ideas within MOC simulations. This is further aided by the new flexible structure detailed in this chapter. The goal of OpenMOC is to promote an active reactor physics community where transparent research is possible and new ideas encourage the improvement of nuclear reactor modeling and simulation.

Chapter 4

Track Laydown

In Chapter 2 the concept of laying down tracks in the MOC method was briefly introduced. In this chapter, the strategies for laying down tracks are discussed in great detail. First, this chapter discusses the relatively straightforward 2D track laydown. Then, the 2D track laydown is used to form a 3D track laydown. This chapter highlights the importance of 3D track laydown decisions on the overall computational cost of MOC, closely following the track laydown analysis of Shaner, et al [18].

4.1 2D Track Generation

As presented in Chapter 2, MOC lays tracks across the geometry over which the MOC equations are applied. While one dimensional, each track represents a particular three dimensional volume and angular subspace. The first step in setting up the MOC problem is creating tracks that span the spatial and angular space of the problem.

When creating the tracks, it is important to consider the boundary conditions as these determine whether the outgoing angular flux needs to be passed as the incoming flux to another track. While analysis of full-core problems typically involves only vacuum boundaries, it is helpful to have the option for reflective boundaries to compare the results of small-core benchmarks, such as the C5G7 benchmark [24], to other codes. For flexibility, OpenMOC focuses on generating tracks that are cyclic and can therefore be used for reflective, periodic, or vacuum boundary conditions.

In this discussion, it is important to make clear the distinction between tracks and segments (sometimes also referred to as intersections). Tracks are defined to span an entire geometry or geometry sub-domain and pass through region boundaries. On the other hand, segments do not cross region or material boundaries.

Tracks for a 2D MOC problem are typically laid down using the cyclic tracking approach and a product quadrature. Users input a desired radial ray spacing $\tilde{\delta}_\phi$ and number of azimuthal angles n_ϕ in $[0, 2\pi]$. With this approach, desired azimuthal angles are created to evenly subdivide the angular space such that the i^{th} desired azimuthal angle $\tilde{\phi}_i \in [0, \frac{\pi}{2}]$ is paired with a complementary azimuthal angle $\tilde{\phi}_{\frac{n_\phi}{2}-i-1}$ as described in Eq. 4.1 and Eq. 4.2.

$$\tilde{\phi}_i = \frac{2\pi}{n_\phi}\left(\frac{1}{2}+i\right) \qquad \forall \qquad i = \left[0, \frac{n_\phi}{2}\right) \tag{4.1}$$

$$\tilde{\phi}_{\frac{n_\phi}{2}-i-1} = \pi - \tilde{\phi}_i \qquad \forall \qquad i = \left[0, \frac{n_\phi}{4}\right) \tag{4.2}$$

Other valid angular quadrature sets can also be used [17]. By tracking both forward and backward along a track, the full 2π angular space is covered as shown in Figure 4-1 for four azimuthal angles. Tracks are laid down such that they intersect with a complementary track at the boundaries.

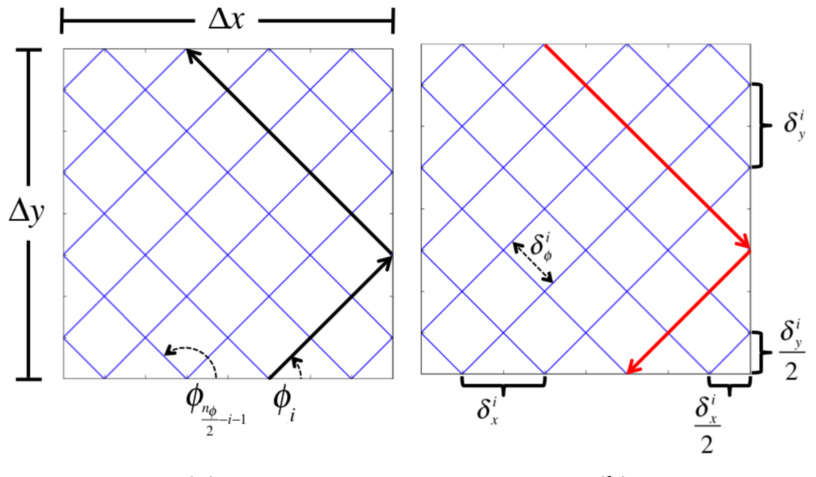


Figure 4-1: Illustration of (a) forward and (b) backward tracking for 2D MOC track laydown. The geometry has dimensions $\Delta x \times \Delta y$.

In order to guarantee cyclic wrapping of 2D tracks, there must be an integer number of tracks on x and y axes for each azimuthal angle. This requires the actual spacing between tracks be chosen as:

$$\delta_x^i = \frac{\Delta x}{n_x^i} \qquad \delta_y^i = \frac{\Delta y}{n_y^i} \tag{4.3}$$

where n_x^i is the integer number of tracks along the x-axis for the i^{th} azimuthal angle. The same notation applies to the y direction. Using the input value of $\tilde{\delta}_\phi$, the integer number of tracks along the axes for the i^{th} azimuthal angle are computed as:

$$n_x^i = \left\lceil \left(\frac{\Delta x \sin \tilde{\phi}_i}{\tilde{\delta}_\phi} \right) \right\rceil \qquad n_y^i = \left\lceil \left(\frac{\Delta y \cos \tilde{\phi}_i}{\tilde{\delta}_\phi} \right) \right\rceil \tag{4.4}$$

where the ceiling is taken to ensure at least one track intersects with each axis. The azimuthal angle also needs to be corrected as:

$$\phi_i = \tan^{-1}\left(\frac{\delta_y^i}{\delta_x^i} \right) \tag{4.5}$$

where ϕ_i is used to denote the i^{th} corrected azimuthal angle. Using the corrected azimuthal angle, the radial ray spacing for each angle is also corrected as:

$$\delta_\phi^i = \delta_x^i \sin \phi_i \tag{4.6}$$

where δ_ϕ^i is used to denote the corrected radial ray spacing. Using the corrected values, ϕ_i and δ_ϕ^i, the 2D tracks are laid down on the geometry.

4.2 Angular Quadrature

In any MOC implementation, angular quadratures are needed. An angular quadrature is a set of angles and weights that can accurately integrate the angular space. In this thesis, a product quadrature is assumed which allows azimuthal and polar quadratures to be separable. For a given azimuthal and polar angle combination, the weight from the azimuthal quadrature is multiplied with the weight from the polar quadrature to determine the total angular weight.

4.2.1 Azimuthal Quadrature

For the azimuthal space, equal angle separation is chosen since it is expected that all azimuthal directions are equally important, as given in Eq. 4.1. The angles need to be corrected to ensure track linking at reflective and periodic boundaries. Azimuthal weights w_ϕ^i for azimuthal index $i = 1...n_\phi/2$ are then chosen as:

$$w_\phi^i = \frac{\phi_{i+1} - \phi_{i-1}}{2\pi} \tag{4.7}$$

Virtual angles are introduced at the extremes for notational convenience. These angles, ϕ_0 and $\phi_{n_\phi/2+1}$, are defined to be:

$$\phi_0 = -\phi_1 \qquad \phi_{n_\phi/2+1} = \pi - \phi_{n_\phi/2} \tag{4.8}$$

The computed weights account for the angular space represented by each track. Note that these relationships define weights for azimuthal angles $\phi \in [0, \pi]$, only accounting for half the azimuthal angular space, since tracking both forward and backward along tracks accounts for the full 2π azimuthal space.

4.2.2 Polar Quadrature

In 2D, the axial dimension is assumed infinite and therefore linking is not required beyond the radial plane. This allows any polar angle quadrature to be chosen without having to correct polar angles. One interpretation is that tracks in 2D MOC are allowed to continue unbounded axially, with their z-height insubstantial, since behavior must be the same along the axial direction due to symmetry. Popular quadratures include the Gauss-Legendre polar quadrature and the TY quadrature [52]. Both are important polar quadrature sets for MOC solvers.

The Gauss-Legendre quadrature set is based on the Gaussian quadrature rule which allows polynomials of order up to $2n-1$ to be integrated exactly using only n quadrature points over the interval [-1, 1]. This is accomplished by choosing quadrature points x_j and weights w_{gl}^j such that

$$\int_{-1}^{1} dx\, P_m(x) = \sum_{j=1}^{n} x_j w_{gl}^j \qquad \forall m = [0, 2n-1] \tag{4.9}$$

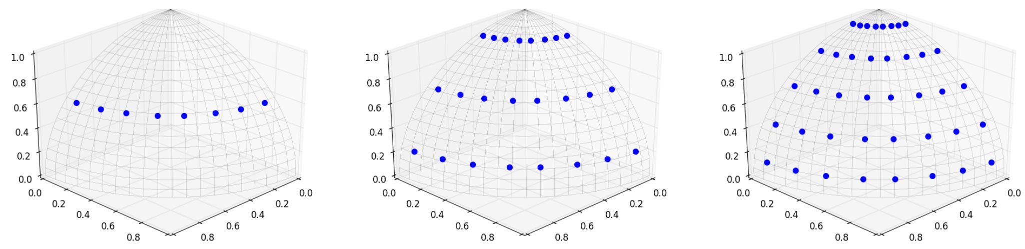
where $P_m(x)$ is the Legendre polynomial of order m. The quadrature points x_j are chosen to be the roots of the Legendre polynomial $P_n(x)$ and the weights w_{gl}^j are computed as

$$w_{gl}^j = \frac{2(1-x_j^2)}{(n+1)^2\left[P_{n+1}(x_j)\right]^2} \tag{4.10}$$

Since the polar angle θ is bounded by $[0,\pi]$, and therefore $\cos\theta$ is bounded by $[-1,1]$, this quadrature rule can be used by the taking polar angles to be $\cos^{-1} x_j$.

Another useful quadrature for 2D MOC, and the most widely used in production MOC codes, is the TY polar quadrature [52]. This quadrature set notes the relation of MOC to collision probability methods, allowing an analytic form for MOC scalar fluxes in terms of collision probabilities. These collision probabilities can then be cast in terms of MOC equations along a strip associated with a 2D track to show that the polar quadrature is optimal when it minimizes the error of approximating Bickley functions. With this insight, the TY quadrature is computationally formed by minimizing the error to Bickley functions, which can be numerically evaluated. In practice, most simulation tools hard-code the published polar angle quadrature points and associated polar weights. While the TY quadrature has been quite successful in 2D simulations, it relies on an analytic form for the neutron transport over a 2D strip. Therefore, the Gauss-Legendre quadrature is typically used in 3D MOC simulations.

Together, the azimuthal and polar quadrature combine to form a product quadrature. An example of a quadrature for 3D MOC is shown in Figure 4-2 where the product quadrature is plotted on a octant of the unit sphere using the Gauss-Legendre polar quadrature.

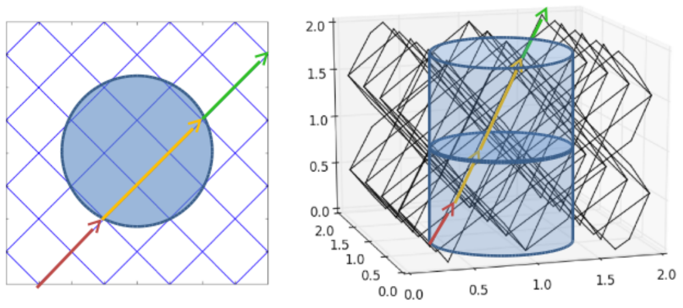


(a) **(b)** **(c)**

Figure 4-2: Illustration of a product quadrature for a 3D MOC problem on an octant of the unit sphere with the Gauss Legendre polar quadrature using 32 azimuthal angles and (a) 2 polar angles, (b) 6 polar angles, and (c) 10 polar angles.

4.3 3D Track Generation

Once the 2D tracks are laid down, 3D tracks can be formed from the 2D tracks. In generating 3D tracks, it is assumed that 3D tracks are laid down as z-stacked groups of tracks that project down onto the 2D track layout. A sparse 2D and associated 3D track laydown for a simple pin-cell is illustrated in Fig 4-3.



(a) **(b)**

Figure 4-3: Illustration of (a) 2D and (b) 3D tracks for a simple pin-cell.

4.3.1 Requirements for Cyclic Track Laydown in 3D

Before discussing the conditions required to generate cyclic tracks in 3D, it is important to understand the concept of a track cycle. Figure 4-4 shows a track laydown in 2D for reflective track cycles with $T^i_{R,k}$ denoting the k^{th} reflective track cycle for the i^{th} azimuthal angle.

To generate 3D tracks, users input a desired track axial ray spacing $\tilde{\delta}_z$ and number of polar angles n_θ in $[0, \pi]$, as well as the parameters required to generate 2D tracks. With

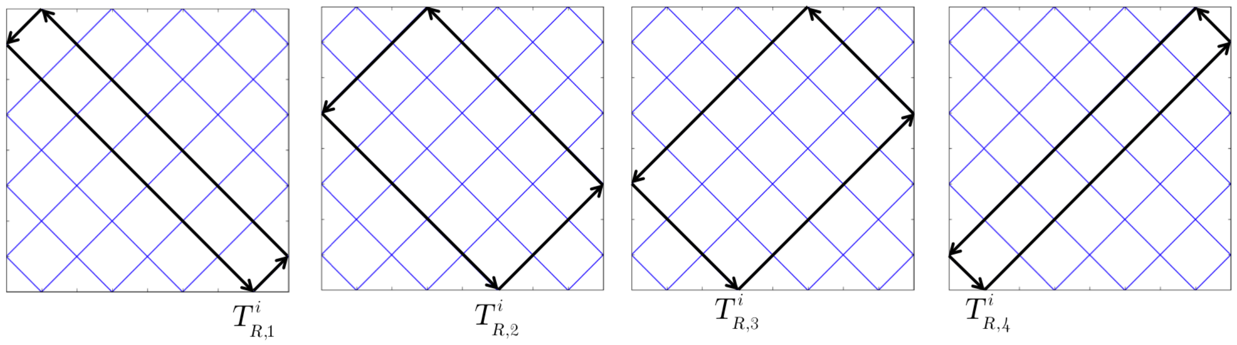


Figure 4-4: An illustration of track cycles. Each plot highlights one of the 2D track cycles labeled $T^i_{R,k}$ denoting the k^{th} track cycle for the i^{th} azimuthal angle.

this approach, desired polar angles are created according to the chosen polar quadrature set. In OpenMOC, it is assumed that quadrature sets are symmetric such that the desired polar angle $\tilde{\theta}_j$ with polar angle index j follows

$$\tilde{\theta}_j = \pi - \tilde{\theta}_{n_\theta - j - 1} \qquad \forall \qquad j = \left[0, \frac{n_\theta}{2}\right), \tag{4.11}$$

where $\tilde{\theta}_{n_\theta - j - 1}$ is the complement of the desired polar angle. Most useful quadrature sets, such as the Gauss Legendre quadrature set, are indeed symmetric.

Note that the polar angles are initially defined to be independent of azimuthal angle index. Later, the corrected polar angle will be dependent on the azimuthal angle to ensure the tracks are cyclic. Following the same notation used to describe the azimuthal angles, the corrected polar angles will be denoted by $\theta_{i,j}$ for azimuthal index i and polar index j. By tracking both forward and backward along a track, the full 4π angular space can be covered, as previously shown in Figure 4-3 for four azimuthal and two polar angles.

Tracks are laid down such that they intersect with a complementary track at the boundaries. Selecting an arbitrary cycle $T^i_{R,k}$ a set of 3D tracks are followed as they complete one 2D cycle. Figure 4-5 highlights a particular 2D track cycle and a set of 3D tracks projected along that cycle.

To guarantee cyclic track wrapping of the 3D tracks, two conditions must be met:

1. The distance between the beginning and end of a 3D track projection along a 2D track cycle must be an integer number of track spacings for each 3D cycle.

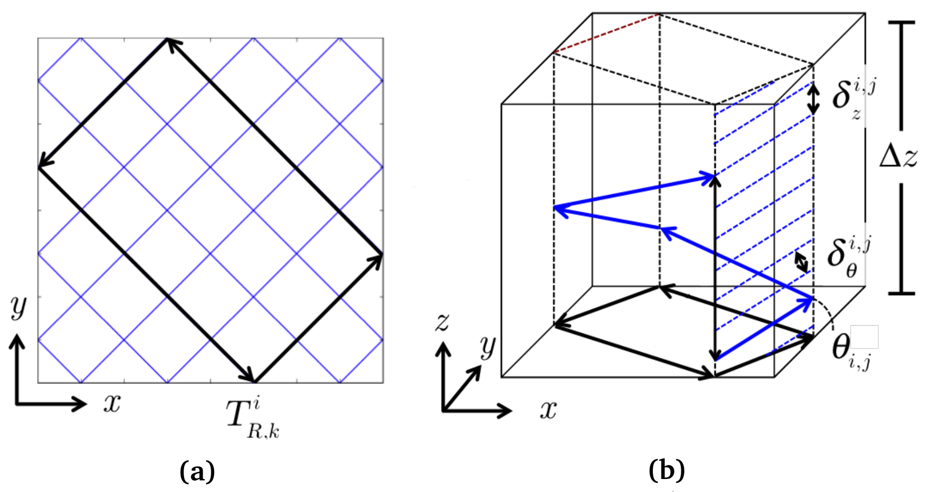


Figure 4-5: Illustration of (a) an arbitrary 2D track cycle $T^i_{R,k}$ and (b) a set of 3D tracks projected along the 2D track cycle. The geometry height is Δz, the track z-spacing is $\delta^{i,j}_z$, and the perpendicular axial distance between tracks is $\delta^{i,j}_\theta$.

2. There must be an integer number of track spacings along the z-axis over the depth of geometry, Δz for each azimuthal/polar angle combination.

The first condition guarantees that a 3D track cycle wraps back onto another 3D track when the 2D reflective cycle is completed. The second condition guarantees that a 3D track cycle that contains a reflection off a top or bottom surface still reflects into an existing 3D track when the 2D cycle is completed. All cyclic 3D track generation schemes must comply with these conditions, though many make additional assumptions.

4.3.2 The Modular Ray Tracing Method

Modular Ray Tracing (MRT) is a popular method for generating tracks, as tracks from different modules naturally link at boundaries. Liu presented the first thorough description of this method for 3D MOC [17] based on a similar approach introduced by Filippone for 2D MOC [53]. MRT uses the previously stated conditions for 3D cyclic ray tracing to generate tracks for a rectangular geometry. After 2D track information has been generated, the 3D track information can be computed using the requirements for 3D cyclic tracking. MRT relies on the principle that a geometry can be uniformly decomposed into a series of rectangular cuboids of equal dimensions, which are referred to as sub-domains. For a sub-domain, the number of tracks and spacing between tracks in x and y are described in Eq. 4.12 and Eq. 4.13, respectively, as

$$n_x^i = \left\lceil \frac{\Delta x \sin \tilde{\phi}_i}{D_x \tilde{\delta}_\phi} \right\rceil \qquad n_y^i = \left\lceil \frac{\Delta y \cos \tilde{\phi}_i}{D_y \tilde{\delta}_\phi} \right\rceil \tag{4.12}$$

$$\delta_x^i = \frac{\Delta x}{D_x n_x^i} \qquad \delta_y^i = \frac{\Delta y}{D_y n_y^i}, \tag{4.13}$$

where D_x and D_y are the integer number of sub-domains in the x and y directions, respectively. This guarantees that an integer number of tracks lie along the x and y boundaries of each sub-domain cell and that tracks on one surface line up with adjoining tracks in the neighbor sub-domain cell.

When generating tracks using the MRT method, it is important to remember that a track crossing a sub-domain interface needs to connect with another track on the neighboring sub-domain. Note that connecting tracks on modular sub-domains are periodic in nature. This allows tracks at sub-domain boundaries to be naturally linked through periodic track linking.

A periodic track cycle is defined to be the series of sub-domain tracks that repeats when a global track traverses a geometry. Figure 4-6 shows the periodic track cycles for one azimuthal angle in our sample geometry when it is split into four sub-domains.

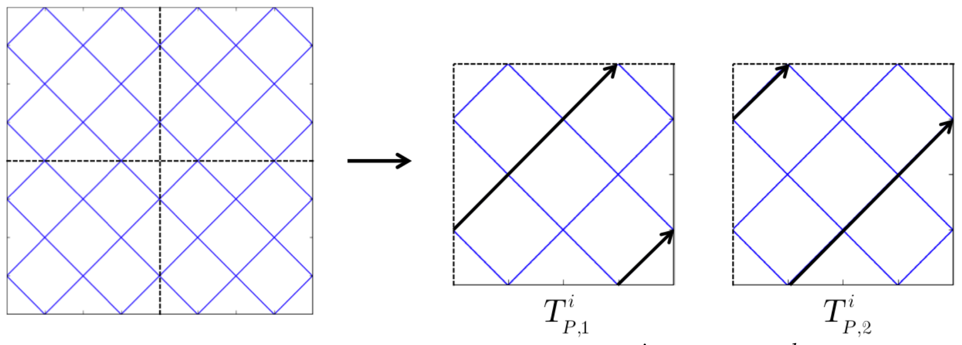


Figure 4-6: An illustration of 2D periodic track cycles $T^i_{P,k}$ for the k^{th} periodic track cycle with the azimuthal index i in a sample geometry split into four sub-domains.

Note that all 2D track cycles for a particular azimuthal angle index i have the same periodic cycle length L_P^i that can be computed as

$$L_P^i = \frac{\delta_x^i}{\cos\phi_i} \text{lcm}\left(n_x^i, \frac{\Delta y}{D_y \delta_x^i \tan\phi_i} \right), \tag{4.14}$$

where lcm is the least common multiple. Using the periodic cycle length for each azimuthal angle, the integer number of track spacings $n_l^{i,j}$ between the beginning and end of a set of 3D tracks after one complete 2D cycle can be computed as

$$n_l^{i,j} = \left\lceil \frac{L_P^i \cot\tilde{\theta}_j}{\tilde{\delta}_z} \right\rceil, \tag{4.15}$$

where the subscript l is used to signify that $n_l^{i,j}$ is measured along the direction of a track cycle. Next, the number of track spacings along the z-axis needs to be set to an integer number. The number of tracks on the z-axis can be found by considering the relation between the number of track spacings in the z-direction and the spacing along the length of the 2D track in Eq. 4.16.

$$\tan\tilde{\theta}_j = \frac{\delta_L^i}{\tilde{\delta}_z} = \frac{L_P^i}{n_l^{i,j}} \frac{n_z^{i,j}}{\Delta z} \tag{4.16}$$

Rearranging and inserting the desired polar angle $\tilde{\theta}_{i,j}$, this equation yields an expression for the number of tracks $n_z^{i,j}$ on the z-axis as

$$n_z^{i,j} = \left\lceil \frac{\Delta z n_l^{i,j} \tan\tilde{\theta}_{i,j}}{L_P^i} \right\rceil, \tag{4.17}$$

where the ceiling ensures at least one track crossing on the z-axis. The corrected z-spacing $\delta_z^{i,j}$ between 3D tracks is computed in Eq. 4.18.

$$\delta_z^{i,j} = \frac{\Delta z}{n_z^{i,j}} \tag{4.18}$$

Using the 2D cycle length and number of track crossings on the z-axis and along the length of the 2D cycle, the polar angle can be corrected using Eq. 4.19.

$$\theta_{i,j} = \tan^{-1}\left(\frac{L_P^i}{n_l^{i,j}\delta_z^{i,j}}\right) \tag{4.19}$$

It is important to note that when polar angles are adjusted, the Gauss-Legendre quadrature set no longer is guaranteed to exactly integrate Legendre polynomials of order up to $2n-1$ with n quadrature points. In fact, it is not guaranteed to exactly integrate any Legendre polynomials beyond the zeroth order. This could either be ignored, assuming that the quadrature can still integrate the polar angle space with reasonable accuracy, or the weights can be re-computed to ensure that Legendre polynomials of order n are exactly integrated. Both options are implemented in OpenMOC, but all the results presented in this thesis ignore the weight correction for simplicity. The angular track weight α_t of track t with azimuthal angle index i and polar index j using a Gauss Legendre quadrature can be computed as

$$\alpha_t = w_\phi^i w_{gl}^j. \tag{4.20}$$

The procedure for identifying the periodic track cycles and finding the start and end points for all tracks using the MRT method can seem a bit tedious, so others have simplified the MRT method by noticing that the lengths of all 2D tracks are an integer multiple of the shortest 2D track [20]. For example, the length of the first three 2D tracks, t_1^i, t_2^i, and t_3^i, for azimuthal index i are shown in Eq. 4.21, Eq. 4.22, and Eq. 4.23.

$$l_1^i = \sqrt{\left[\frac{\delta_x^i}{2}\right]^2 + \left[\frac{\delta_y^i}{2}\right]^2} = \frac{1}{2}\sqrt{(\delta_x^i)^2 + (\delta_y^i)^2} \tag{4.21}$$

$$l_2^i = \sqrt{\left[\frac{3\delta_x^i}{2}\right]^2 + \left[\frac{3\delta_y^i}{2}\right]^2} = \frac{3}{2}\sqrt{(\delta_x^i)^2 + (\delta_y^i)^2} = 3l_1^i \tag{4.22}$$

$$l_3^i = \sqrt{\left[\frac{5\delta_x^i}{2}\right]^2 + \left[\frac{5\delta_y^i}{2}\right]^2} = \frac{5}{2}\sqrt{(\delta_x^i)^2 + (\delta_y^i)^2} = 5l_1^i \tag{4.23}$$

Since the length of all 2D tracks is an integer number of lengths of the first track, l_1^i, any valid track laydown for the first track is also valid for all other tracks. Therefore, the cycle length L_P^i for azimuthal index i can be set to the length of the shortest track, l_1^i. This simplified form is termed Simplified Modular Ray Tracing (s-MRT). The algorithm for generating tracks for the MRT and the s-MRT method can be described by Alg. 4-1.

It is important to note that the track generation procedure described in Alg. 4-1 favors correcting the axial ray spacing over correcting the polar angle. Alternative algorithms could easily be designed to favor correcting the polar angle over the axial ray spacing, but it is expected that correcting the axial ray spacing results in less induced error and this is in line with previous work on MRT [20].

When the periodic cycle length is small relative to the desired axial ray spacing, the correction to the axial ray spacing can be very large. This has significant implications for the s-MRT method where the periodic cycle length is always on the order of the radial ray spacing. Shaner showed that for realistic 3D MOC problems where the axial ray spacing can be quite coarser than the radial ray spacing, s-MRT introduces far more tracks than necessary [18]. This was shown to increase the number of tracks generated by at least an order of magnitude for common full-core PWR problems. Since the computational cost of MOC scales directly with the number of tracks, an order of magnitude increase in the number of tracks translates directly into an order of magnitude increase in the run time.

4.4 OpenMOC Implementation

Due to the great flexibility of modular ray tracing methods and the run-time concerns of s-MRT, MRT was chosen as the track generation technique for OpenMOC. In earlier development stages, multiple track generation options existed, but since MRT outperformed all options in both flexibility and efficiency, the other options were eliminated.

In addition to implementing MRT for 3D Track Generation, OpenMOC generates tracks on-the-fly rather than pre-computing and saving all track data. With the large number of tracks involved in a 3D calculation, the storage of 3D track information can be expensive. Therefore, 3D tracks are computed on-the-fly with track indexes. The track

Algorithm 4-1: 3D track generation using the Modular Ray Tracing Method

```
User specifies $n_\phi$, $\tilde{\delta}_\phi$, $n_\theta$, $\tilde{\delta}_\theta$, $D_x$, $D_y$, and $D_z$.
for all $i \in I$ do                                        ▷ Loop over all azimuthal angles
    Compute desired azimuthal angle $\tilde{\phi}_i$
    Compute the # of tracks in $x$ and $y$ directions, $n_x^i$ and $n_y^i$ (Eq. 4.12)
    Compute radial spacings between tracks $\delta_x^i$ and $\delta_y^i$ (4.13).
    Correct the azimuthal angle and ray spacing, $\phi_i$ and $\delta_\phi^i$ (Eq. 4.3 and Eq. 4.4).
    if MRT then
        Compute the length of the 2D periodic track cycles, $L_P^i$ (Eq. 4.14).
    else
        Compute the length of the shortest track, $l_1^i$ (Eq. 4.21).
        Set $L_P^i \leftarrow l_1^i$.
    end if
    for all $j \in J$ do                                    ▷ Loop over all polar angles
        Compute the # of track spacings after one complete 2D cycle, $n_l^{i,j}$.
            $$n_l^{i,j} = \left\lceil \frac{L_P^i \cot \tilde{\theta}_j}{\tilde{\delta}_z} \right\rceil$$
        Compute the # of tracks on the z axis, $n_z^{i,j}$.
            $$n_z^{i,j} = \left\lceil \frac{\Delta z n_l^{i,j} \tan \tilde{\theta}_{i,j}}{L_P^i} \right\rceil$$
        Compute the z-distance between 3D tracks, $\delta_z^{i,j}$.
            $$\delta_z^{i,j} = \frac{\Delta z}{D_z n_z^{i,j}}$$
        Correct the polar angle, $\theta_{i,j}$.
            $$\theta_{i,j} = \tan^{-1}\left(\frac{L_P^i}{n_l^{i,j} \delta_z^{i,j}}\right)$$
    end for
end for
```

indexes used to identify a track are: the azimuthal angle index, the xy track number, the polar angle index, and the axial track number. As a pre-processing step, auxiliary information is saved, such as the actual ray spacings, the actual polar angles, the 2D track information, and the number of tracks in a cycle. Since retrieving track data is simple given this auxiliary information, track retrieval adds trivial computational cost to the simulation.

4.5 Conclusion

Efficiently laying down tracks is important to any MOC solver. In this chapter, the methodology for laying down 3D tracks to guarantee cyclic relationships at boundaries was discussed in detail. The MRT track laydown scheme was chosen for its flexibility. Under this scheme, tracks can link at domain boundaries through periodic conditions. More importantly, corrected ray parameters do not deviate significantly from desired ray parameters. This is especially important for typical 3D reactor physics problems in which the radial plane contains much more complexity than the axial direction, allowing for a potentially coarser ray spacing in the axial direction than radial direction which MRT can easily accommodate. Other track laydown schemes, such as s-MRT, have great difficulties in accommodating these conditions, leading to significantly more tracks being generated than needed, and therefore directly leading to a higher computational cost.

Chapter 5

Ray Tracing

In the previous chapter, track generation was discussed. Once the tracks are laid across the geometry, they need to be partitioned by source region intersections into segments across which the MOC equations can be applied. At a minimum, the segment lengths and associated source region identifiers need to be recorded for each track. During the transport sweep, the MOC equations described in Chapter 2 are applied to all segments. This chapter discusses ray tracing procedures in which the segments are formed from the track and geometry information.

This chapter introduces a ray tracing approach for axially extruded geometries in which ray tracing is performed in a 2D/1D fashion for efficiency. It is important to note the distinction between a 2D/1D approach to ray tracing and a 2D/1D approach to MOC. In this thesis, all problems are solved using 3D MOC but the ray tracing approach splits radial and axial detail to efficiently form 3D segments.

5.1 Introduction to Ray Tracing

Ray tracing fundamentally solves the problem of determining the next intersection with a surface along a direction of travel and which new region is entered after the intersection. Since complicated geometries contain many surfaces, this can be expensive if all surfaces in the geometry need to be tested for intersections. Much of the work in complex ray tracing, often for animations, relies on somehow forming *acceleration structures* which limit the number of surfaces that need to be tested [54]. Traditional methods form acceleration structures that resemble some hierarchy of cells and surfaces grouped by location so that only nearby surfaces are considered [55].

In traditional reactor physics problems, an acceleration structure is naturally formed by the hierarchical structure of the geometry. For instance, a core can largely be viewed as an arrangement of assemblies, each of which is comprised of an arrangement of fuel rods. Therefore, by just using the hierarchical CSG form of the geometry, the ray tracing requirements are naturally reduced.

However, ray tracing can still be expensive, especially for cases which do not form an easily structured form, such as reactor baffles, grid spacers, and neutron shields. A ray tracing algorithm which is capable of handling all of these complexities becomes significantly more complicated, leading to a loss in computational efficiency. In moving from 2D simulations to 3D simulations, even more surfaces are introduced, exacerbating the issue.

Most MOC implementations avoid this issue by treating ray tracing as a pre-processing step whereby the segment information, which includes its length and the ID of the traversed source region (SR), is calculated and explicitly stored at the beginning of the simulation. During the transport sweeps, the information is then loaded from memory. In this way, the amount of repeated ray tracing work is reduced and the cost of ray tracing is effectively amortized over the number of transport sweeps. While this approach is straightforward, its memory and compute requirements for 3D MOC can be prohibitive, even for small problems, due to the vast number of segments present in 3D MOC simulations [56]. Reducing the memory footprint is important for many reasons including improved cache efficiency and reducing bulk memory requirements.

In this thesis, an alternative approach is presented that greatly reduces the segment storage and generation requirements by taking advantage of the extruded geometry structure common to many reactor physics problems. This alternative approach saves no 3D segment data, rather treating the ray tracing problem as a coupled 2D and 1D system whereby 2D radial ray tracing information is combined with 1D axial information to compute the 3D intersections [56].

Another ray tracing scheme, termed the CCM [25], also sought to reduce the memory requirements for ray tracing axially extruded geometries by only saving the unique chords (or segments). However, this method still requires that the segmentation procedure be performed for all 3D tracks prior to performing transport sweeps, which can be prohibitively expensive for complex geometries. In addition, while it does indeed reduce the storage requirements, the storage scales with the number of unique 3D segment lengths. This can be problematic for problems in which there are a large number of unique segments.

The implementation in this thesis aims to most efficiently solve typical PWR problems. From experience in simulating single PWR assemblies, the axial ray separation can be on the same order as the axial source height [57], potentially causing there to be a large number of unique segments. This motivates using concepts from the CCM method to identify repeated segment lengths but also relying on the 2D/1D ray tracing approach to reduce storage.

5.2 Forming an Axially Extruded Geometry

As previously stated, OpenMOC only supports piece-wise axially extruded geometries. Most common reactor problems are naturally piece-wise axially extruded so this is not a strong limitation in practice. Here, an axially extruded geometry is defined to be a geometry in which every radial plane in the geometry is identical whereas a piece-wise axially extruded geometry can be defined as the collection of a finite number of axial zones where the geometry over each axial zone constitutes an extruded geometry.

For ray tracing in a 2D/1D scheme, a single axially extruded geometry is required. Therefore, all radial detail must be gathered in order to effectively convert the piece-wise axially extruded geometry to an axially extruded geometry. While common reactor cores contain variations from axially extruded geometries, such as end plugs for control rods, these variations can be fully captured by implicitly inserting additional geometric intersections. Note the axially extruded requirement is only applied to the geometry, not the materials. Each axial zone in an axially extruded region could be a different material.

Given an axially extruded geometry, it is possible to store only the 2D segments associated with intersections in the radial geometry. The 3D segments can then be formed on-the-fly using a simple axial mesh.

First, 2D segments need to be formed which reflect intersections of the 2D tracks with a superposition of all radial surfaces in the geometry. This is done by simultaneously ray tracing across all the unique radial planes in the geometry, as depicted in Fig. 5-1. This creates an implicit geometry containing all radial information, termed the *superposition plane*. Each 2D geometric region in the superposition plane corresponds to an axially extruded region with an associated unique identifier that contains an axial mesh and an array of associated 3D SRs.

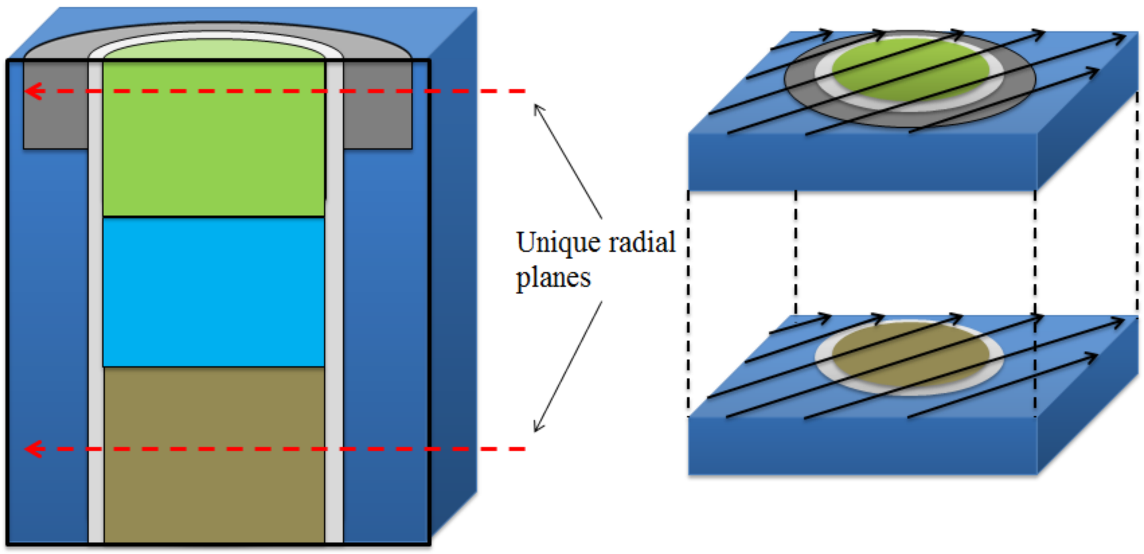


Figure 5-1: Depiction of 2D ray tracing for superposition of all radial detail.

Each 2D segment formed during the ray tracing contains its length and the unique identifier of the axially extruded region being traversed. After 2D segmentation, axial meshes need to be created for on-the-fly axial ray tracing. If a global axial mesh is desired, whereby all axially extruded regions have the same axial mesh, all the unique z-planes in the geometry are collected and sorted into a single axial mesh. Otherwise, local meshes are populated for each axially extruded region during initialization of the 3D SRs. To initialize the 3D SRs, a temporary vertical ray is created for each axially extruded region. These rays are all upward directed, starting from the bottom of the root geometry, and segmented to determine distances between axial intersections while initializing the associated SRs. During this step, the location of a point within each SR is noted, and the data structures associated with managing the SRs are initialized. An illustration of this process is presented in Fig. 5-2.

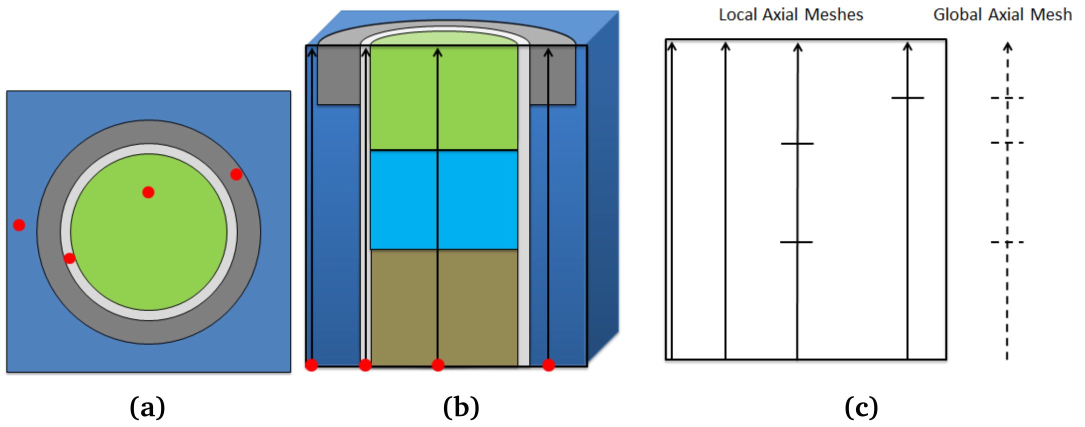


Figure 5-2: Illustration of SR initialization starting from the superposition plane (a) where a radial point is chosen in each unique region, vertical tracks (b) are generated at the chosen points to map the axial detail, yielding (c) the 1D axial meshes for each region, and (optionally) a global 1D axial mesh

Implicitly, this strategy can create extra radial intersections since some of the axial levels might not have originally contained the full radial detail of the superposition plane. However, the number of additional intersections should be low due to the regular structure of most reactor cores. For instance, fuel rods with end plugs present only a slight deviation from an axially extruded geometry.

The advantage of local axial meshes is the ability to have different axial refinements within the reactor. For instance, a partially inserted control rod might require a finer SR discretization near the control rod tip. If a global axial mesh were used, the finer discretization would need to be applied to the whole geometry at that axial height. With local axial meshes, only the regions which need refinement would use a finer discretization.

5.3 On-the-fly Axial Ray Tracing

During the transport sweeps, all 3D tracks are traversed across their span of the geometry. The common method mentioned in Section 5.1, in which segments are formed in a pre-processing step, accomplishes this by splitting every 3D track into 3D segments before the transport sweeps and then simply cycling through all the segments.

In the methods presented here, 3D segments are recreated on-the-fly using information from the 2D ray trace over all radial detail and the 1D axial meshes, both formed at the beginning of the simulation. Due to the manner in which the 3D tracks were generated, as detailed in Chapter 4, each 3D track has a corresponding 2D segmented track. From the starting z coordinate of a track along with its polar angle, it is possible to determine the distance along the track to both intersections with axial planes and intersections with radial surfaces, defined by the 2D segments. On-the-fly axial ray tracing can either be performed on each 3D track individually or on an entire z-stack.

5.3.1 Ray Tracing Individual 3D Tracks

For ray tracing each 3D track individually, the associated 2D segments are traversed until the 3D track reaches its endpoint. First, the starting point is used to determine the appropriate starting 2D segment and index into the axial mesh. If local axial meshes are used, the index needs to be recomputed with a binary search at the start of each 2D segment. If a global mesh is used, the axial index only needs to be calculated at the beginning of the track. The 2D segments are traversed and the shorter distance to either an axial or radial intersection is calculated. This computed distance is the 3D segment length and has an associated 3D SR indicated by the axial index. To form the next 3D segment, the position along the 2D track is moved by the appropriate distance. This process is repeated until the endpoint is reached. An illustration of this concept is presented in Fig. 5-3.

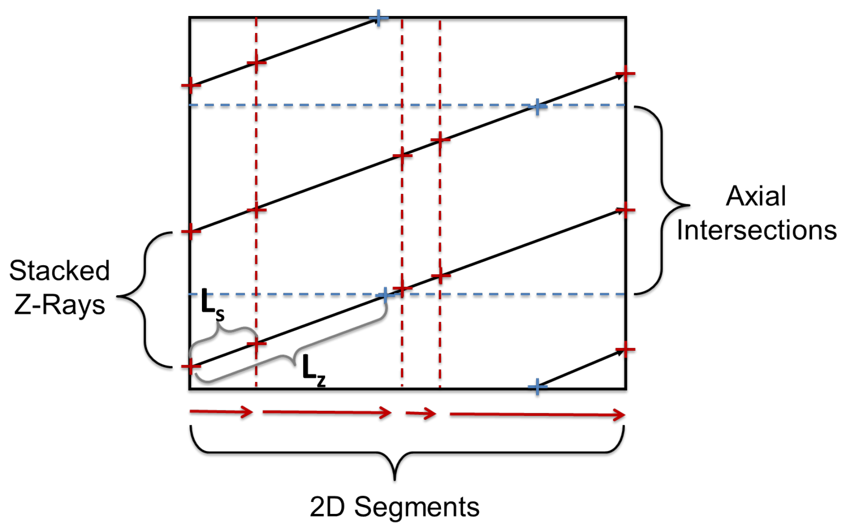


Figure 5-3: Illustration of the on-the-fly axial ray tracing process with axial intersections colored in blue and radial intersections colored in red. For the chosen track, the distance to the next axial intersection is denoted L_z and the the distance to the next radial intersection is denoted L_s.

5.3.2 Ray Tracing 3D Track z-Stacks

The structure of the 3D track laydown can be used to ray trace an entire z-stack. Specifically all tracks in the z-stack have the same polar angle θ, project onto the same 2D track, and are separated by a constant axial ray spacing δz. Therefore, the axial height z_i of the i^{th} lowest track (starting from 0) can be given as a function of distance s along the associated 2D track as

$$z_i(s) = z_0(0) + i\delta z + s \cot\theta \tag{5.1}$$

where $z_0(0)$ is the z-coordinate at the intersection of the lowest track with the z-axis at the start of the associated 2D track. Combining this with 2D track information is enough to completely describe the trajectory and location of 3D tracks in the stack. Therefore, it is possible to determine which tracks will traverse a given SR.

For each 2D segment in the 2D track, there is an associated axially extruded 2D region which contains a list of 3D SRs in the region. With this structure, the SRs can be traversed rather than the tracks. Using the boundaries of the SR and Eq. 5.1, it is possible to analytically compute the indexes in the z-stack of tracks that will cross the

SR as

$$i_{start} = \left\lceil \frac{z_{min} - \max\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z} \right\rceil \tag{5.2}$$

$$i_{end} = \left\lfloor \frac{z_{max} - \min\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z} \right\rfloor \tag{5.3}$$

where i_{start} is the index of the first track to cross the SR, i_{end} is the index of the last track to cross the SR, s_{start} is the 2D distance traversed at the start of the segment and s_{end} is the 2D distance traversed at the end of the segment. A detailed derivation of these relationships can be found in Appendix D. A depiction of this process is shown in Fig. 5-4.

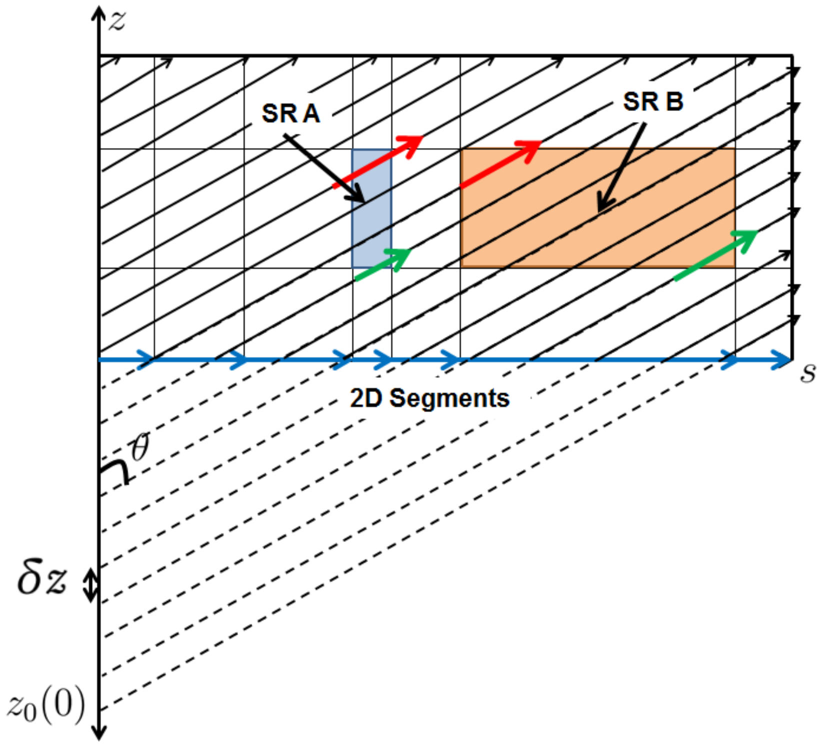


Figure 5-4: Illustration of the on-the-fly axial ray tracing process for an entire z-stack. The green arrows denote the first track to traverse the highlighted SRs calculated by Eq. 5.2 and the red arrows denote the last tracks to traverse the SRs calculated by Eq. 5.3. In SR A, a group of tracks traverse the entire 2D segment length. In SR B, one track traverses the entire axial source height.

Notice that for SR A depicted in Fig. 5-4 there are multiple track segments that cross the entire 2D length of the SR and are therefore identical. Their 3D segment length L_{3D}

would simply be

$$L_{3D} = \frac{s_{end} - s_{start}}{\sin\theta}. \tag{5.4}$$

Due to the radial direction having much greater complexity than the axial direction, SRs will tend to be longer in the axial direction than in the radial plane, especially when higher order sources allow for coarse discretization in the axial direction. The high aspect ratio causes some tracks to cross the entire 2D length of the SR.

To take advantage of the repeated lengths, it is possible to calculate the indexes of the first and last tracks to traverse the entire 2D segment length. For this condition to be met, the axial height of the tracks over the entire 2D segment must be greater than the minimum axial boundary of the SR and less than the maximum axial boundary. Therefore, the indexes of interest are the first track to have its lowest point above the minimum boundary and the first track to have its highest point above the maximum boundary. These indexes, i_{in} and i_{out}, respectively, can be calculated as:

$$i_{in} = \left\lceil \frac{z_{min} - \min\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z} \right\rceil \tag{5.5}$$

$$i_{out} = \left\lceil \frac{z_{max} - \max\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z} \right\rceil \tag{5.6}$$

A derivation of these relations can be found in Appendix D. For each SR these indexes are calculated along with the beginning and end track indexes given in Eq. 5.2 and Eq. 5.3. It is possible that i_{in} will be greater than i_{out} when the polar angle is steep enough, allowing for the SR to be fully traversed axially without traversing the entire 2D length. These tracks can be determined with indexes i_{in} and i_{out} and have a common 3D segment length L_{3D} given by

$$L_{3D} = \frac{z_{max} - z_{min}}{|\cos\theta|}. \tag{5.7}$$

Therefore, 3D segments are classified into four categories named by Sciannandrone [25]:

- **Mixed Tracks – Case A:** Tracks that partially traverse the SR and cross the lower SR boundary. They are defined by track indexes $i_{start} \leq i < \min(i_{in}, i_{out})$. For these tracks, each 3D segment is computed individually.

- **Mixed Tracks – Case B:** Tracks that partially traverse the SR and cross the upper SR boundary and each 3D segment is again computed individually. They are defined by track indexes $\max(i_{in}, i_{out}) \leq i \leq i_{end}$.

- **Horizontal Tracks:** Tracks that fully traverse the entire 2D radial distance of the SR and are defined by indexes $i_{in} \leq i < i_{out}$.

- **Vertical Tracks:** Tracks that traverse the entire axial distance of the SR and are defined by $i_{out} \leq i < i_{in}$.

This classification of segments allows the computational work involved in calculating track intersections to be greatly reduced. In addition, the horizontal and vertical tracks for each SR have common segment lengths, allowing for the potential reduction in computational work. While not implemented for this thesis, these common segment lengths all have the same exponential term for the MOC equations, allowing these terms to only be computed once for the group of equal length segments.

Another important distinction between this scheme and ray tracing by individual 3D track is the traversal order of segments. Since the ray tracing by z-stack scheme describes intersections of all tracks with a given region, segments are traversed by *region* rather than by track. In this scheme, the algorithm loops over all regions which are intersected by the z-stack, treating all segments which intersect the region before moving to the next region. This has profound implications on cache performance and is the subject of the next section.

5.4 Performance Considerations

Now that the on-the-fly ray tracing algorithms have been defined, performance considerations on typical computer hardware are discussed. This section focuses on how these algorithms interact with the hardware design of computers in order to maximize performance.

5.4.1 Cache Considerations for Segment Traversal

In the context of ray tracing, an efficient algorithm should traverse regions in a similar order in which the data is organized. This means information relating to SRs, including the neutron source and scalar fluxes, should also be allocated contiguously in memory according to the traversed order. However, complex geometric detail in the radial plane causes any ordering to be very difficult. For instance, a completely optimized ordering of source regions in one direction might be a completely sub-optimal ordering in the perpendicular direction. This is illustrated in Figure 5-5 for a simplified pin-cell. As the geometric complexity and number of regions grows, the stride between SRs along the perpendicular track would also grow, leading to a cache miss on almost every new source region along the sub-optimal track.

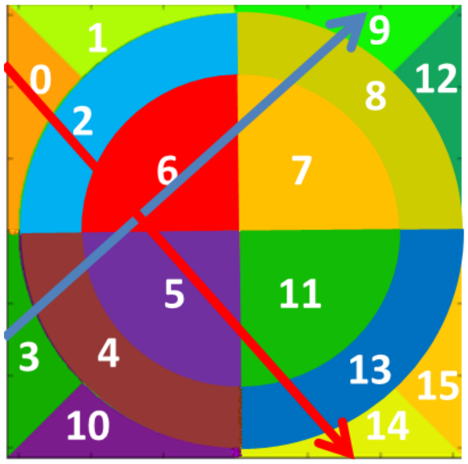


Figure 5-5: A depiction of optimizing 2D radial SR ordering for the track highlighted in blue. A perpendicular track, highlighted in red, experiences a very sub-optimal SR ordering over its traversal with larger strides between sequential SRs.

However, the axial geometric detail for axially extruded geometries is far more regular. This allows for the ordering of regions in the axial direction to be organized, leading to improved cache performance when operating on nearby regions with the same 2D projection. This ordering is illustrated in Figure 5-6. Note that this ordering is naturally created by the SR initialization scheme discussed in Section 5.2.

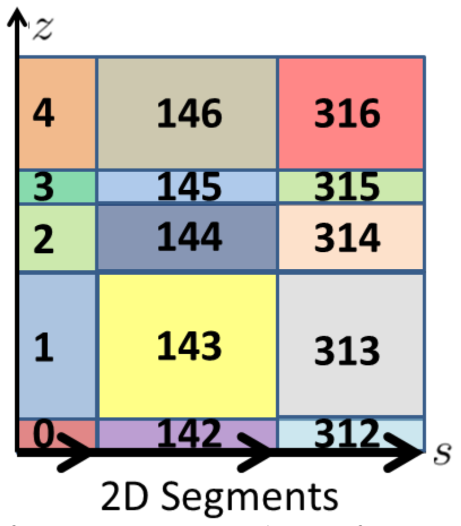


Figure 5-6: A depiction of optimizing SR ordering for vertical intersections. SRs are ordered sequentially in the axial direction, causing all vertical intersections to only have a stride of one SR (the stride of scalar flux, neutron source, and moment information) in memory.

For ray tracing by individual 3D tracks, this memory layout means that intersections with vertical surfaces cause a likely cache hit whereas intersections with horizontal surfaces, dictated by the 2D segments, likely incur cache misses. For ray tracing by z-stack, segments are traversed by SR rather than track. In this scheme, SRs are traversed in order axially with all segments of a given SR treated together. This greatly improves the locality for loading SR information.

However, the improved locality for loading SR information does come at a cost. When ray tracing individual 3D tracks, all segments correspond to the same track and thus only one set of boundary angular flux data is updated while traversing segments. This means the angular flux data for the track is always kept in a low level cache. When ray tracing an entire z-stack, the segments belong to various tracks. While these tracks are generally close in memory, there is a possibility of cache misses when switching tracks. Due to the nature of the z-stack traversal, the algorithm switches tracks on nearly every traversed segment. When the number of energy groups becomes large, the angular flux data associated with a track also becomes large, and the possibility of incurring cache misses when switching tracks increases.

5.4.2 Temporary Storage of Segments

The 3D segments that are computed from the axial on-the-fly ray tracing can either be directly used to apply the MOC equations on-the-fly or stored in buffers in the order in which they are formed. It is important to remember that in the OpenMOC implementation, as well as many other MOC implementations, each segment represents both a forward and backward angular flux in order to maximize efficiency. Therefore, if segments are stored in buffers, the ray tracing cost can be halved since the ray tracing operation would not need to be repeated in the reverse direction. With the segments stored in buffers, the buffer can be traversed forward and backward, representing forward and backward angular fluxes.

Another benefit of storing the segments in a buffer is the ability to group together similar operations. All segments for a selected track or z-stack are ray traced, yielding a buffer of segments. Afterwards, the MOC equations are applied to all segments in the buffer. This allows for improved locality since the two tasks are separated and nearby data is used more frequently during each task.

The cost of storing segments in buffers is the additional memory cost. The memory for the buffer is allocated as an array at the beginning of the solver. Since each thread handles different segments simultaneously, each tread needs its own buffer. The size of each buffer should be sufficient to store the computed segments. For ray tracing by track, this is equal to the maximum number of segments for a single track. For ray tracing by z-stack, it is equal to the maximum number of segments for an entire z-stack. The memory required for the maximum segments per track is trivial in comparison with the total memory footprint of the solver since there are many tracks in the problem and buffers only need to be created for each thread. The memory cost of the buffers is higher when ray tracing by z-stack, but still very small in comparison with the overall memory footprint. However, the need to store information into buffers could lead to detrimental latency effects when loading the segment information.

5.5 Results

Since there are trade-offs between the different ray tracing schemes, they should be analyzed within the context of full core 3D MOC simulations. Previous work studied these schemes on a portion of a single C5G7 assembly [56]. It is important to note that the geometry had the same axial discretization in each radial region such that using a global axial mesh incurs no penalty. The results showed that:

- Storing segment information in temporary buffers outperforms on-the-fly application of the MOC equations
- Ray tracing by stack was more efficient than ray tracing by track for all configurations
- Using a global axial mesh did not improve performance over local axial meshes for ray tracing by z-stack
- Using a global axial mesh was more efficient than local axial meshes for ray tracing by individual 3D track
- All on-the-fly ray tracing schemes significantly outperformed explicit storage of segment information in storage requirements with practically no computational overhead

From these results, temporary storage of segment information was chosen for the final OpenMOC implementation and used for the calculation of all results in this thesis. Additionally, explicit storage of segment information as a preprocessing step was not considered. Most of these results are expected to hold for a wide variety of problems.

The aim of this thesis is to analyze full core problems with a high number of energy groups (70) whereas the C5G7 benchmark uses 7-group cross-section data. Higher group counts have the potential to significantly change the performance characteristics as ray tracing costs become trivial in comparison with work done applying the MOC equations. Therefore, the cost of using local axial meshes should be trivial in comparison with other costs, motivating the use of local axial meshes in all cases. However, the order in which segments are traversed could still have a substantial impact on performance.

5.5.1 Simulation Parameters

The results presented here concentrate on the differences between ray tracing by individual 3D track and ray tracing by z-stack with both schemes using local axial meshes. The SDSA test problem, described in Appendix E.2.5, is simulated with the simulation parameters given in Table 5.1, which resemble expected MOC ray spacing and mesh parameters for the linear source solver. Note that no ring divisions are used in this study. For all computational results, each case is run three times, reporting the mean and estimating the error margin using the maximum deviation from the mean.

Table 5.1: MOC parameters for the SDSA test problem for ray tracing studies

Number of Sectors in Moderator	8
Number of Sectors in Guide Tubes	8
Number of Sectors in Fuel	4
Height of Flat Source Regions	2.0 cm
Radial Ray Spacing	0.05 cm
Axial Ray Spacing	0.75 cm
Number of Azimuthal Angles	64
Number of Polar Angles	10
Number of Transport Sweeps	1

5.5.2 Single Thread Performance Comparison

First, the single thread performance of the two on-the-fly ray tracing schemes is compared. Results using one node of the Falcon supercomputer are presented in Table 5.2, showing ray tracing by z-stack to have slightly better single-threaded performance than ray tracing by individual track. Both schemes result in integration times near 10 ns for the flat source approximation. The linear source approximation adds a factor of 2×–3× the runtime for the fixed mesh, but recall that linear source allows for a coarser mesh to be used. In these trials, each test is run three times. The reported uncertainty indicates the maximum deviation from the average value.

Table 5.2: Single thread performance of ray tracing schemes using one node of the Falcon supercomputer

Source Approx.	Ray Tracing Scheme	Transport Sweep Time (s)	Integration Time (ns)
Flat	By Track	923.4 +/- 0.7	12.2 +/- 0.01
Flat	By z-Stack	820.6 +/- 0.8	10.8 +/- 0.01
Linear	By Track	2385.9 +/- 6.1	31.41 +/- 0.08
Linear	By z-Stack	2268.7 +/- 2.1	29.87 +/- 0.03

5.5.3 Parallel Scaling

As noted in Chapter 3, the on-node parallelism of OpenMOC uses OpenMP shared memory parallelism. In this section, the strong scaling performance is considered for both ray tracing schemes in which the number of created threads is varied for the SDSA problem of fixed size. In this analysis, the number of threads never exceeds the number of cores such that cores and threads become synonymous as each core is assigned at most one thread.

To analyze parallel performance, the speedup metric is considered which for n threads is defined by T_1/T_n where T_1 is the single-threaded computation time and T_n is the computation time with n threads. When each core is assigned at most one thread, the ideal speedup with n threads is n. The parallel performance on the SDSA test problem of flat source MOC is presented in Fig. 5-7 for both ray tracing schemes. The same comparison is shown in Figure 5-8 for linear source MOC.

For practical purposes, performance using the full available resources is more important than single thread performance or scalability. Therefore, it is important to compare the runtime of the algorithms using the maximum number of available cores on a single node. This comparison is shown in Table 5.3, showing that ray tracing by z-stack seems to have slightly better performance than ray tracing by individual 3D track for both flat and linear source approximations of the neutron source.

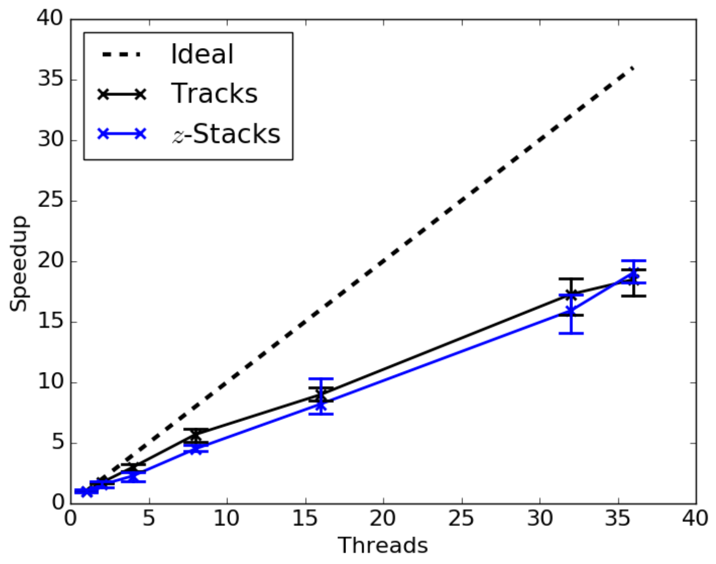


Figure 5-7: Strong scaling performance on the SDSA test problem using the Falcon supercomputer with on-the-fly ray tracing by tracks and by z-stacks, both using the flat source MOC solver.

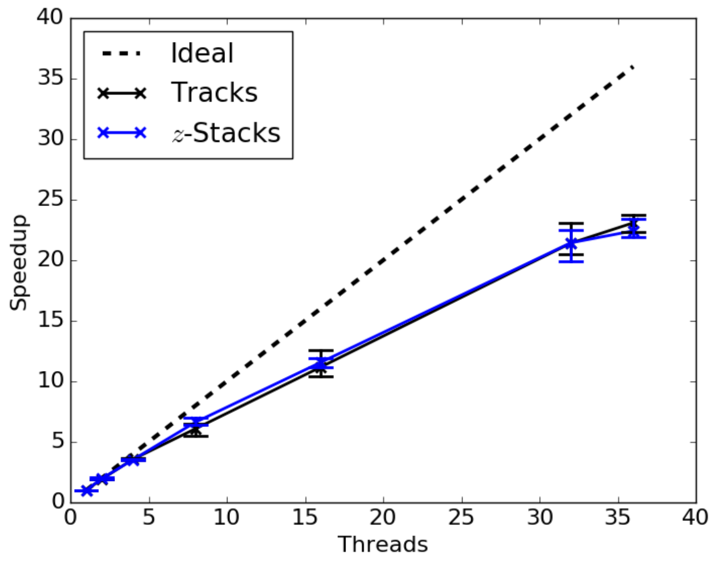


Figure 5-8: Strong scaling performance on the SDSA test problem using the Falcon supercomputer with on-the-fly ray tracing by tracks and by z-stacks, both using the linear source MOC solver.

5.5.4 Performance on Cetus

Since the aim of this thesis is accurately simulating full core PWR problems, which requires immense computational resources, the performance should be compared on

Table 5.3: Performance on the SDSA test problem using full computational resources of a single Falcon node with 36 cores

Source Approx.	Ray Tracing Scheme	Transport Sweep Time (s)	Integration Time (ns)
Flat	By Track	49.9 +/- 3.9	23.8 +/- 1.8
Flat	By z-Stack	43.1 +/- 2.3	20.5 +/- 1.1
Linear	By Track	103.3 +/- 3.4	49.0 +/- 1.8
Linear	By z-Stack	101.2 +/- 4.4	47.9 +/- 2.2

a machine capable of solving very large problems. This work targets the Argonne BlueGene/Q supercomputer. Since the architecture of this machine is very different from traditional nodes, with comparatively slow cores and the ability to host four hyper-threads per core, the performance of algorithms on this machine should also be compared.

For testing performance on the Argonne BlueGene/Q supercomputer, the Cetus partition is utilized. One limitation in terms of testing performance on the Cetus partition is a one hour runtime limit of all jobs. Because of this limitation, the MOC ray parameters are significantly coarsened to those shown in Table 5.4 so the problem can run on one thread in under an hour.

Table 5.4: MOC ray parameters for the SDSA test problem for ray tracing studies on the Cetus partition of the Argonne BlueGene/Q supercomputer

Radial Ray Spacing	0.1 cm
Axial Ray Spacing	1.5 cm
Number of Azimuthal Angles	16
Number of Polar Angles	6

First, the performance is tested using the full computational resources of a single node of the Cetus partition. Since each node features 16 cores with 4 hyper-threads per core, this means using 64 threads. The results are shown in Table 5.5.

From these results, it seems that yet again the on-the-fly ray tracing by z-stack performs slightly better than on-the-fly ray tracing by individual 3D track. It is important to note the slowdown in integration time from Falcon to Cetus, highlighting the extent

Table 5.5: Performance on the SDSA test problem using full computational resources of a single Cetus node with 16 cores

Source Approx.	Ray Tracing Scheme	Transport Sweep Time (s)	Integration Time (ns)
Flat	By Track	13.83 +/- 0.06	76.3 +/- 0.3
Flat	By z-Stack	13.52 +/- 0.04	74.6 +/- 0.2
Linear	By Track	31.88 +/- 0.52	176.0 +/- 2.9
Linear	By z-Stack	31.68 +/- 0.26	174.9 +/- 1.4

to which the cores on Cetus are slower than the cores on Falcon.

Since the scaling studies on Falcon showed scaling to be quite similar between ray tracing methods, only the scaling using on-the-fly ray tracing by z-stack is analyzed on Cetus.

It is also important to note the role of hyper-threads in scaling studies. Since hyper-threads share computational resources with other hyper-threads on the same core, ideal scaling is only tangible when the number of threads is less than the number of cores. In this case, the number of cores is 16 and the parallel speedup of the algorithm is 91.9% of ideal for flat sources and 90.9% for linear sources when one thread is used per core, which is significantly better than the scaling observed on Falcon. At 64 threads, where all available hyper-threads are utilized, the average speedup is 42.15 for flat sources and 37.8 for linear sources. This shows that the algorithm benefits greatly from the addition of hyper-threads, showing that there latency is a major contributor to the overall run-time. The full scaling results are plotted in Figure 5-9.

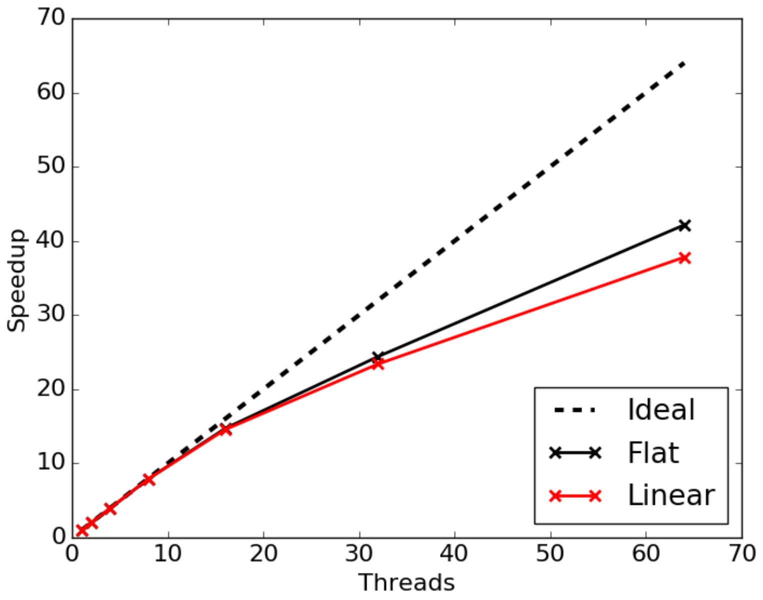


Figure 5-9: Strong scaling performance on the SDSA test problem using the Cetus partition of the Argonne BlueGene/Q supercomputer with on-the-fly ray tracing by z-stacks.

5.6 Conclusion

In this chapter, the MOC ray tracing problem was thoroughly discussed. Alternative approaches to ray tracing for 3D MOC were presented whereby only 2D segments are stored and 3D segments are computed on-the-fly. Two approaches were analyzed: one in which each individual 3D track is ray traced and another where an entire z-stack of tracks is ray traced. Both approaches offer significant memory reduction with minimal or no computational overhead. Results comparing these two on-the-fly ray tracing algorithms show that the ray tracing by entire z-stacks of 3D tracks was most efficient. Parallel scaling of the algorithms was also studied, showing similar scaling between the two ray tracing algorithms. Parallel scaling was quite close to ideal linear scaling for the targeted Argonne BlueGene/Q architecture, even into the hyper-thread regime. This supports the expectation that OpenMOC should make efficient use of the BlueGene/Q computational resources for full core simulations.

Highlights

- On-the-fly ray tracing is chosen to alleviate the computational burden of explicitly storing an immense number of 3D segments
- All 2D radial geometric detail forms a superposition plane which is used in conjunction with axial meshes to form an axially extruded geometry
- On-the-fly ray can either be accomplished for each individual track or with an entire z-stack of tracks together assuming constant axial ray spacing
- For a high number of energy groups, differences in performance of ray tracing schemes are mostly dependent on the memory access patterns for traversing source regions
- Results show the on-the-fly ray tracing by z-stack to be slightly preferable to on-the-fly ray tracing by individual track for the 70-group energy structure used in this thesis
- The parallel scaling of the flat source solver using on-the-fly ray tracing methods achieves near-linear over threads for the targeted Argonne BG/Q architecture

Chapter 6

Domain Decomposition

In the previous chapters, the on-node shared memory parallelism of OpenMOC was discussed in which data is shared and available to all threads on a single computational node. However, this thesis concentrates on solving large scale reactor physics problems which cannot be solved with just one computational node. When extending to multiple computational nodes, communication of data between nodes becomes extremely costly. This makes a shared parallelism model computationally infeasible across nodes. Therefore, hybrid parallelism is introduced in which on-node parallelism uses the OpenMP shared parallelism model, but MPI [14] is used to communicate information across nodes. Specifically, spatial domain decomposition is implemented in which each MPI process is responsible for the work over a geometrical sub-domain.

6.1 Geometrical Decomposition

There are a wide variety of options for decomposing a problem into sub-domains, each of which are assigned to a single MPI process. In this thesis, spatial domain decomposition is selected for both its simple interpretation and scalability. In spatial domain decomposition, the geometry is partitioned into many rectangular parallelepiped sub-domains. Each MPI process (usually one per node) is assigned one of the geometrical sub-domains and is responsible for simulating the neutron behavior over the region. A 2D depiction is given in Figure 6-1.

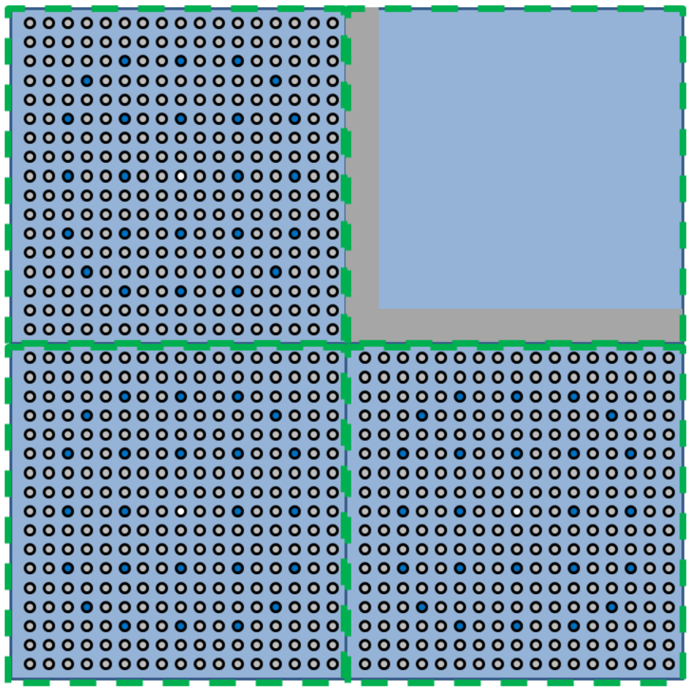


Figure 6-1: An illustration of partitioning a geometry into sub-domains. The partition, shown in green, forms a 2×2 lattice of sub-domains.

Each geometrical sub-domain can be approached as a somewhat independent reactor physics problem. However, the MOC equations require estimates of the incoming boundary angular fluxes. Since these angular fluxes are carried along tracks, this requirement can be satisfied by linking tracks at sub-domain boundaries and communicating the angular flux information. To enforce the linking of tracks, the MRT track laydown algorithm is implemented, as discussed in Chapter 4, with each sub-domain required to be of identical dimensions and track laydown. Under the MRT scheme, tracks automatically link at reflective and periodic boundaries. By ensuring periodic track linking, tracks naturally meet at sub-domain boundaries. This is illustrated in Figure 6-2.

It is important to note that while the track laydown across each sub-domain is identical, the internal geometry of each sub-domain can be different. Therefore, ray tracing must be conducted on each sub-domain separately, using the techniques described in Chapter 5. Each sub-domain forms a separate superposition of radial detail. Since only the local sub-domain superposition plane is used during ray tracing, the extra intersections formed from transforming piece-wise axially extruded geometries to pure axially extruded geometries (discussed in Chapter 5) can be reduced as the number of

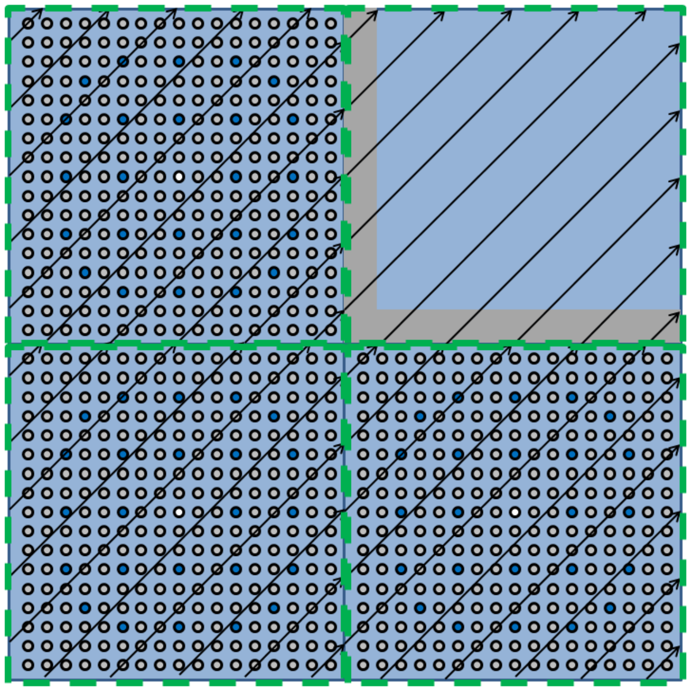


Figure 6-2: An illustration of track-liking with an MRT track laydown. Tracks are shown for just one direction. Notice that the track laydown on every domain is the same and the linking track can be found by examining the periodic connecting track on the domain.

axial sub-domain partitions increases.

6.2 MPI Communication

In any domain decomposition scheme, it is important to identify the data that needs to be communicated between sub-domains. The communication between nodes in OpenMOC is entirely handled by MPI, which is an industry standard for inter-node communication in scientific applications. An MPI process is defined as the series of programmed instructions which can be managed independently by an operating system scheduler [58]. Each MPI process is responsible for exactly one sub-domain. In the context of this thesis, one MPI process is used per node. However, it is possible to run multiple MPI processes per node with the node splitting its computational resources between its assigned MPI processes. Since the domain decomposition algorithm in this thesis is designed with the expectation of one MPI process per node, communication will be discussed as being between nodes rather than MPI processes.

The communicated quantities in this thesis fall into two categories: boundary quantities and global quantities. Boundary quantities only need to be communicated between the neighboring nodes whereas global quantities need to be communicated between all nodes. Since communication costs scale with the number of communicating nodes, communication of global quantities is far more costly than the communication of boundary quantities. Therefore, an optimal domain decomposition algorithm should keep the data associated with global quantities small, such as scalar values. Once the communication data is identified, algorithms can be formed to transmit the data.

6.2.1 MPI Fundamentals

The fundamental MPI communication protocols are sends and receives, which can either be blocking or non-blocking. During blocking sends and receives, if node *A* sends data with an `MPI_Send` function to node *B*, it will wait until node *B* calls a corresponding `MPI_Recv` function to receive data from node *A*. Likewise, node *B* will wait until it finds the matching send from node *A*. Once there is a match, node *A* sends the data to node *B* at the specified memory addresses and once the communication is complete both nodes *A* and *B* then continue their operations.

In contrast, non-blocking sends and receives do not wait for the matching send / receive functions to continue. Instead, the send and receive are merely posted and when they match the data is transmitted. When using non-blocking sends and receives it is important to manage the send and receive statuses with care in order to ensure that the data is actually received before use. Therefore, the non blocking MPI send and receive functions (`MPI_Isend` and `MPI_Irecv`, respectively) are also created with an `MPI_Request` object which monitors the communication. The object can be queried using the `MPI_Test` function to determine whether the data transfer has been completed. In general, non-blocking communication can be much faster than blocking communication since processes are not required to wait but requires extra book-keeping to monitor the status of MPI messages.

In addition to standard send and receive messages, MPI also contains functions for global reductions. A *reduction* is an operation applied to data across many nodes. A global reduction is a reduction over all nodes. An example of this is a summation of values across all nodes, though other operations exist such as maximum and minimum. For instance, consider a domain decomposed problem in which the total neutron production rate is desired. To compute this quantity, each node could first compute the local neutron production on its sub-domain. A reduction can be used to then sum the local neutron production rates to form the total neutron production rate.

Global reductions come in both blocking and non-blocking forms. The blocking and non-blocking reduction functions are `MPI_Allreduce` and `MPI_Iallreduce`, respectively. Since reductions in OpenMOC are usually implemented on scalar data rather than vectors, the communicated data is small, so the cost of the communication is relatively small. Therefore, blocking communication is always chosen for reductions in OpenMOC for simplicity and for guaranteeing synchronization across all nodes during reductions.

6.2.2 The Buffered Synchronous Algorithm

A common theme in the MPI communication algorithms for both the MOC and CMFD solvers is the need to communicate information with nodes of neighboring sub-domains. Since most of the communication in OpenMOC falls into this category, it is important to use an algorithm that is efficient. Here, an efficient algorithm for communication with any collection of neighbors is presented. The concept is to use separate buffers for transferring data with each neighbor with non-blocking communication. All send and receive messages are posted in a non-blocking fashion. Then, each node waits for all its sends and receives to complete before unpacking the data. This process is described in detail in Algorithm 6-1 from the perspective of a single node communicating with its neighbors.

It is important to underscore that this algorithm works for any collection of neighbors a given node might have, which do not need to be adjacent. For application in OpenMOC, this algorithm is applied to spatially adjacent neighbors. Boundary communication for angular flux data is limited to face-adjacent and corner-adjacent neighbors whereas

Algorithm 6-1: Buffered Synchronous algorithm for transferring information with neighboring nodes

```
Consider a node with neighbors $U$
Send buffer $S_u$ and receive buffer $R_u$ have been initialized for each neighbor $u$
for all $u \in U$ do                              ▷ Loop over all buffers for neighboring nodes
    Fill send buffer $S_u$ with information to transfer to node $u$
end for
for all $u \in U$ do                                        ▷ Loop over all neighboring nodes
    Post non-blocking send message $M_{\text{send},u}$ to $u$ with data from buffer $S_u$
    Post non-blocking receive message $M_{\text{receive},u}$ to $u$, which will fill buffer $R_u$
end for
$A \leftarrow$ true                              ▷ $A$ indicates whether communication is active
while $A$ do
    $A \leftarrow$ false
    for all $u \in U$ do                         ▷ Loop over all messages to neighboring nodes
        if $M_{\text{send},u}$ is active or $M_{\text{receive},u}$ is active then
            $A \leftarrow$ true
        end if
    end for
end while
for all $u \in U$ do                              ▷ Loop over all buffers for neighboring nodes
    Copy data from buffer $R_u$ into local data structures
end for
```

CMFD boundary data is only communicated with face-adjacent neighbors.

A major advantage of the Buffered Synchronous Algorithm over blocking communication is the ability for a single MPI process to communicate with multiple neighbors at once. Since nodes often have physical connections with more than one other node, it is important to make full use of all the available connections in order to maximize efficiency.

6.3 MOC Inter-domain Communication

6.3.1 Identification of Communicated Quantities

For MOC, there is relatively little data that needs to be communicated between nodes, since each sub-domain can be approached as a nearly stand-alone reactor physics problem. Most of the communication deals with boundary angular flux data. With the tracks linked at sub-domain boundaries, each node should communicate the outgoing boundary flux information for its tracks with neighboring sub-domains. Ignoring boundary sub-domains, which might not need to communicate information across outer boundary surfaces, the number of boundary angular fluxes each node needs to communicate with its neighbors is $2TG$ where T is the number of tracks per sub-domain, G is the number of MOC energy groups, and the factor of two arises from angular fluxes existing in both the forward and reverse directions of each track.

While the boundary angular flux communication accounts for the vast majority of the MOC communication, a few global quantities need to also be calculated. First, residuals are required to evaluate convergence criteria. Second, total reaction rates are necessary to form estimates of the eigenvalue k when no CMFD acceleration is present. This is computed with the fission, leakage, and absorption rates. In addition, reaction rates are necessary to normalize the MOC scalar fluxes, which are normalized by the total fission source. Since all of these global quantities are scalar values, they add very little to the overall inter-node communication costs of OpenMOC.

6.3.2 Communication Algorithm

Since the bulk of the MOC communication costs relate to the communication of boundary angular fluxes, the algorithm for transferring boundary angular flux information is described in great detail here. In order to simplify the algorithms for communicating angular flux data, a bulk synchronous communication scheme is chosen whereby all angular fluxes are communicated after the transport sweep, rather than during the transport sweep. This implies that angular fluxes at domain interfaces are lagged, which might slow convergence. This effect is studied later in Section 8.6, as part of the MOC convergence sensitivity studies presented in Chapter 8.

Before and after the communication step, there are *synchronization barriers* which force all nodes to wait until all other nodes have reached the barrier before continuing. This ensures that data is not overwritten that is necessary for the transport sweeps. In addition, it allows for simple recording of the run-time spent in the MOC communication stage.

Determining connecting tracks is important for communicating angular flux data. Recall that the MRT track laydown scheme allows for each node to compute the connecting tracks on neighboring sub-domains since the track laydown is identical on all sub-domains. The index of the connecting track on the neighboring sub-domain is simply the index of the periodic track in the current sub-domain. Since tracks run in both forward and reverse directions, the direction of the connecting track is also relevant.

By providing the neighboring node with the track index, the track direction, and the boundary angular flux data, the neighboring node can copy the angular flux data into its local boundary angular flux arrays.

The communication of angular fluxes between nodes can be accomplished by using the Buffered Synchronous algorithm with corner-adjacent neighbors in addition to face-adjacent neighbors since tracks can cross sub-domain corners. However, due to the structure of the MRT track laydown, there are no tracks through xy corners. Therefore, there is a maximum of 14 neighboring sub-domains. Nodes are assigned with an `MPI_Cart` object which groups neighbors together by locality.

In order to use the Buffered Synchronous algorithm, buffers need to be setup that temporarily store the communication data. A naive approach would create buffers capable of storing all boundary angular flux data. While this would accurately communicate the data, it would significantly add to the on-node memory footprint of the algorithm and would also likely have slow communication due to the large size of data being sent at once.

Instead, the Buffered Synchronous algorithm is applied iteratively in which smaller buffers are packed with some angular flux and connecting track data. After the Buffered

Synchronous algorithm completes, new data is packed into the buffers and the process repeats until all boundary angular flux data has been successfully communicated with neighbor domains. Structuring the communication algorithm in this way allows for all communication channels to be filled frequently with significantly lower bandwidth usage per communication round, yielding improved performance. The buffer size chosen in this thesis was large enough to fill the information of 1000 tracks. With 70 group data, this amounts to approximately 1 MB, which was found to be ideal during early prototype tests [49].

The algorithm is given in Algorithm 6-2 which assumes that all tracks link with a neighbor sub-domain. This might not be true for geometry boundaries where there is no neighbor domain, but this can be overcome for notational convenience by assuming these tracks communicate with their own domain where their own domain is a neighboring domain. To simplify the presentation of the algorithm, tracks are assumed to be in only one direction, though abstraction to traversing tracks in both forward and reverse directions is simple.

Algorithm 6-2: MOC boundary angular flux communication algorithm for transferring information with neighboring nodes

```
Consider a node with neighbors $U$
Send buffer $S_u$ and receive buffer $R_u$ have been initialized for each neighbor $u$
Buffers have size $LG$, where $L$ is defined by the user, $G$ is the number of groups
The current node contains $T$ tracks
Initialize vector $V$ of size equal to the number of neighbors $|U|$ with elements $V_u$
$V_u \leftarrow 0 \quad \forall u \in U$

$A \leftarrow$ true
while $A$ do                    ▷ While there are boundary angular fluxes to communicate
    for all $u \in U$ do                              ▷ Loop over all neighboring nodes
        $z \leftarrow 0$
        $t \leftarrow V_u$
        while $t < T$ and $z < L$ do                   ▷ Loop over all un-seen tracks
            if Track $t$ connects with neighbor domain $u$ then
                Place data of size $G$ for track $t$ in buffer $S_u$ at location $zG$
                $z \leftarrow z + 1$
            end if
            $t \leftarrow t + 1$
            $V_u \leftarrow t + 1$
        end while
    end for
    Run the Buffered Synchronous algorithm described in Algorithm 6-1 for $U$ neighbors
    if $V_u = T \quad \forall u \in U$ then
        $A \leftarrow$ false
    end if
end while
```

6.4 CMFD Inter-domain Communication

6.4.1 The CMFD Eigenvalue Solver

Before discussing the domain decomposition implementation of CMFD, the specific algorithm used to solve the CMFD equations should be discussed. The CMFD equations described in Appendix B form a generalized eigenvalue problem in which the elements of the standard matrices can be easily formed. This allows the CMFD system to be solved by more standard solvers than the MOC equations.

While many algorithms exist to solve generalized eigenvalue problems, the CMFD implementation in OpenMOC focuses on power iteration with a red-black SOR algorithm to invert the linear system during every inner iteration. A notable aspect of the red-black SOR algorithm is that all cells are assigned a color in a checkerboard pattern, as illustrated in Figure 6-3.

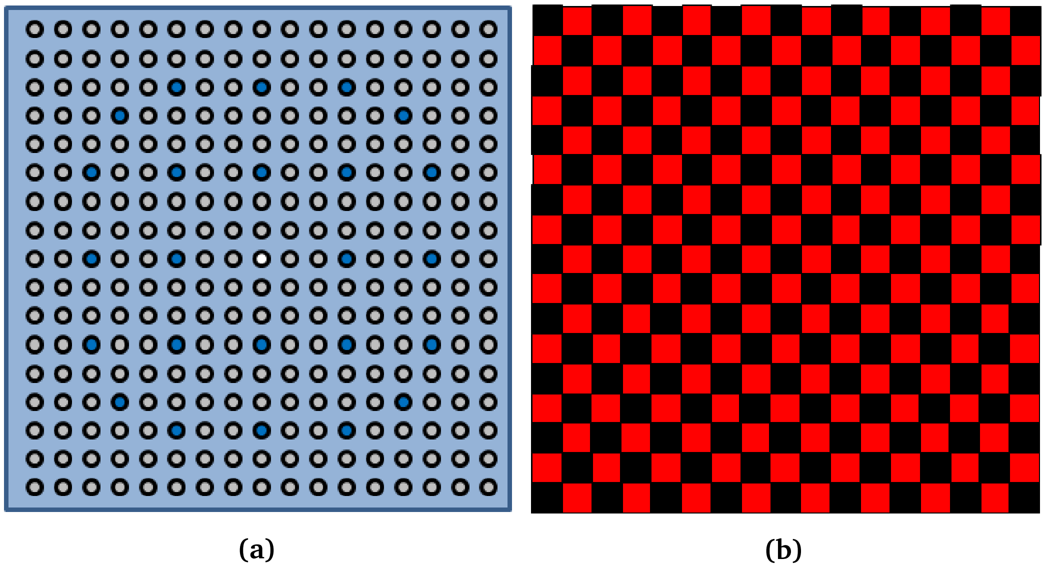


Figure 6-3: A depiction of the red-black cells in the CMFD solver. Here a single assembly geometry (a) is modeled with a uniform CMFD mesh (b) with cells encoded in a red-black checkerboard pattern.

Each iteration is split into two stages: one dealing with the red cells and one dealing with the black cells. Note that the solution in each cell only depends on its neighbors, which are of the opposite color. First, the solution in the red cells is updated with the previous solution on black cells. Then, the solution of the black cells is updated with the new solution in the red cells. In this way, all cells in each stage can be computed in parallel. For structuring the communication algorithm, it is important to note that this scheme only solves for half of the scalar fluxes in each iteration. This means only half the boundary terms need to be communicated at each stage.

6.4.2 Identification of Communicated Quantities

In solving the CMFD equations, the main communication between nodes involves the boundary CMFD *scalar fluxes* at every red/black stage during the red-black SOR algorithm. In addition, boundary CMFD diffusion coefficients, volumes, and surface currents must be communicated at the start of the CMFD eigenvalue solver setup during each transport sweep iteration.

A few global quantities need to be communicated. Similar to MOC, these are limited to residuals and reaction rates. The only difference with CMFD global quantities is their definition. Rather than being defined in terms of MOC scalar fluxes, they are defined in terms of CMFD scalar fluxes.

6.4.3 Communication of Boundary Currents

At the beginning of the formation of the CMFD eigenvalue solver during each transport sweep, some boundary values are communicated. Boundary diffusion coefficients and volumes can be handled easily using the Buffered Synchronous algorithm described in Alg. 6-1. Communicating surface currents is more difficult due to nuances from edge and corner cases. Specifically, the CMFD solver treats currents as existing on CMFD cell faces, decisions need to be made when MOC tracks intersect cell edges (here defined to be xy, xz, or yz corners) or cell vertexes (xyz corners). In OpenMOC, any track within 10^{-12} cm of a corner is treated as a corner crossing. In handling these edges and vertexes, it is critically important that an MOC track's full current arrive in the CMFD cell that the track traverses.

6.4.3.1 Handling Edge and Vertex Currents

Before discussing how OpenMOC treats the communication of CMFD surface currents, the process for handling edge and vertex currents needs to be described. A track is assumed to intersect a surface when it is within 10^{-12} cm of the surface. This leads to the possibility of intersecting cell edges and vertexes. In OpenMOC, currents are directly tallied on cell faces, edges, and vertexes during transport sweeps. This means that each CMFD cell has 26 tally surfaces (6 face surfaces, 12 edge surfaces, and 8 vertex surfaces) in 3D rather than just the 6 face surfaces used in the CMFD calculation. In OpenMOC all currents are defined as leakage from a particular CMFD cell. For instance the current shown in Figure 6-4 would be tallied as a current leaking from the positive x surface of CMFD cell C rather than a current entering cell D along the negative x surface. Therefore, currents entering a cell are gathered from current tallies leaving neighboring cells across opposite surface directions.

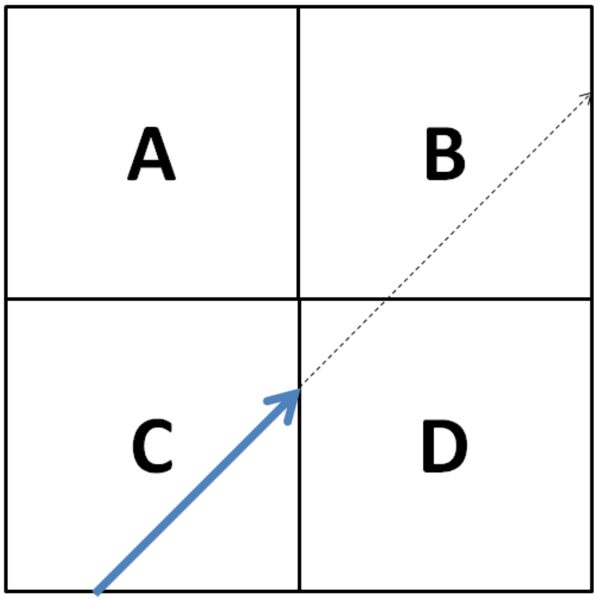


Figure 6-4: An illustration of an MOC track traversing CMFD cells. The current from the blue portion of the MOC track is tallied as an outgoing current on the positive x surface of cell C.

After the transport sweep, currents are split from edges and vertexes onto faces such that the current is delivered to the cell with which the MOC track connects. This is accomplished by having the current traverse across neighboring cells and into the connecting cell.

For edge currents, this is easy to visualize, as shown in Figure 6-5. In this instance the current from the MOC track is split onto the faces that meet at the edge, as shown in red. Each split current receives half the current of the original MOC tallied current. In order for the two split currents to transmit to the connecting CMFD cell, they then traverse the other surface composing the edge through the neighboring cell.

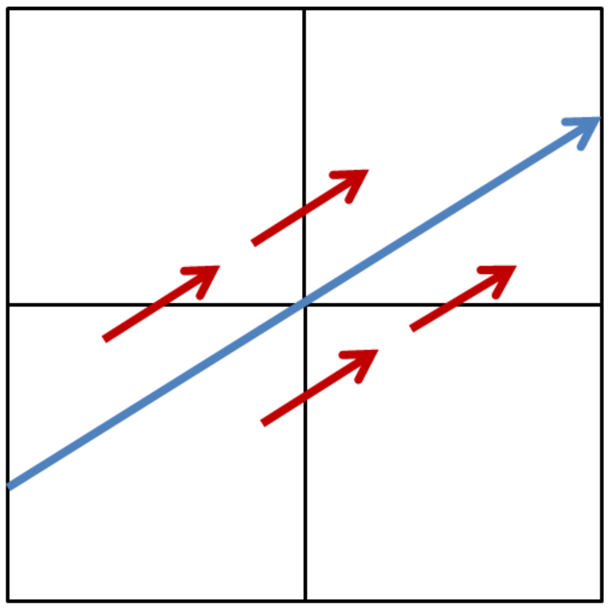


Figure 6-5: Illustration of splitting currents from an MOC track (blue) crossing an edge surface to currents on surface faces (red).

It is important to note that this process involves tallying currents on neighboring CMFD cells in order to properly transmit currents. For vertex currents, a similar approach is taken. However, rather than splitting the vertex current directly onto faces, the current is split onto the three corresponding edges, as shown in Figure 6-6. Since edges comprise to surfaces (such as xy), the remaining face (in this case z) must be crossed in the appropriate neighboring cell in order to properly deliver the current to the connecting cell. For vertex currents, each split current takes one third of the original current since it is split along three edges.

Since the splitting of vertex currents tallies current to edges, the vertex splits must be done before edge splits. Otherwise, the splitting of edge currents would need to be done twice. One way to conceptualize this methodology is by artificially perturbing an MOC track's location by an infinitely small amount such that the imagined virtual track does not cross an edge or corner. The perturbation is applied in all relevant directions (i.e., shifted both positively and negatively in x or y for an xy edge) with equal weight so that no bias is induced.

For each of the virtual tracks, it is important to treat boundaries carefully to consis-

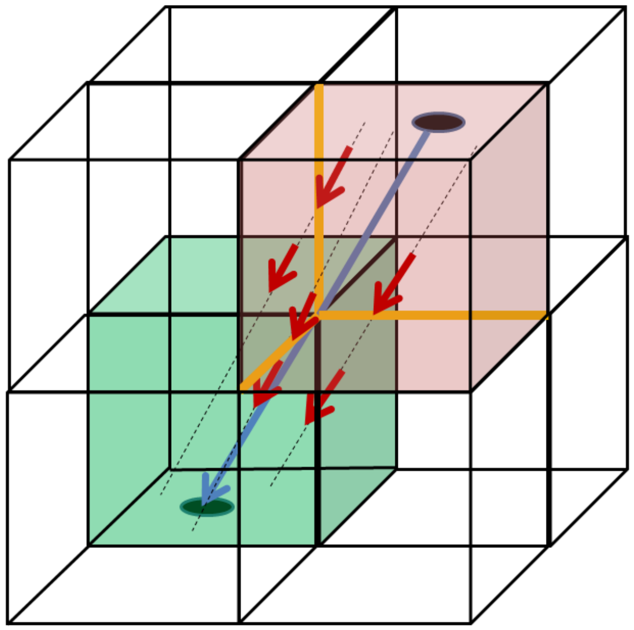


Figure 6-6: Illustration of an MOC track (blue) crossing from the red cell to the green cell through a vertex surface split into currents (red) intersecting edges (highlighted in orange). The currents (red) then continue to the green cell through surface face intersections.

tently capture the effect the virtual track would have on a boundary. For instance, if the virtual track encounters a vacuum boundary, it needs to be counted as current leaking the geometry. However, if it encounters a reflective boundary, the track needs to reflect onto the correct surface. This is illustrated in Figure 6-7.

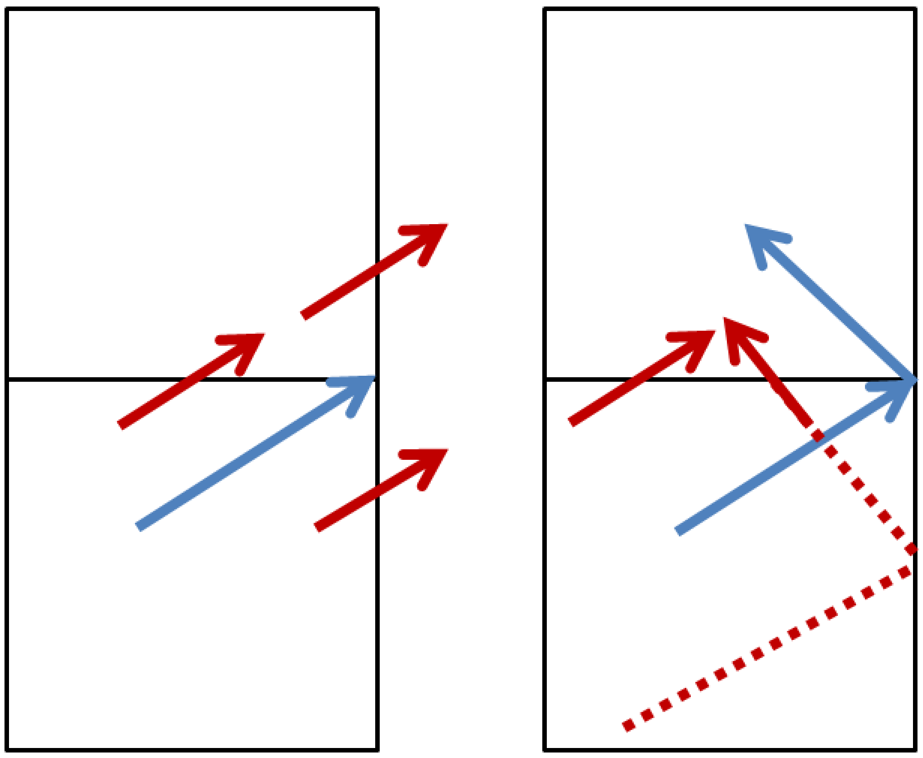

(a) Vacuum (b) Reflective

Figure 6-7: Split currents (red) for an MOC track (blue) intersecting a boundary edge for (a) vacuum boundary conditions and (b) reflective boundary conditions.

6.4.3.2 Communicating Edge and Vertex Currents

Since the process of splitting edge and vertex currents tallies face and edge currents onto neighboring cells, nodes may need to tally currents onto another node's sub-domain. In order to keep the off-domain tallied currents local with the 6 communicating neighbors, the communication is conducted in two steps. First, the vertex currents are split and the corresponding off-domain edge and face currents are communicated. Then, edge currents are split and off-domain face currents are communicated. When these currents are communicated, the off-domain currents are received by the appropriate domain and added to the local tally of the corresponding surface current.

Since vertex currents are split before communication, only the 18 face and edge currents need to be communicated between nodes. The two stage communication process allows current to be transmitted with corner neighbors with only direct communication between the 6 nodes representing face-adjacent sub-domains. Again, this communication can be implemented using the Buffered Synchronous algorithm.

6.4.4 Communication of Boundary Scalar Fluxes

The most important communication component of the CMFD algorithm is the communication of domain boundary scalar fluxes. Since the number of boundary CMFD scalar fluxes on a typical sub-domain are usually quite small in comparison with the MOC boundary angular fluxes, the Buffered Synchronous algorithm can be applied directly to the red-black SOR scheme with face-adjacent neighbors. Since only half the boundary scalar fluxes are communicated in each red/black iteration, the communication buffers can be chosen to be half the size of the number of scalar fluxes on each boundary.

Since the CMFD cells are laid out in a uniform grid and each buffer location corresponds to at most two CMFD cells, the mapping of cells to buffer locations is straightforward, eliminating the need to communicate indexes.

6.5 Results

In order to evaluate the performance of the domain decomposition implementation, both strong scaling and weak scaling studies are conducted. Strong scaling studies analyze the performance as cores are added to solve a problem of fixed size, as we have seen in the past for on-node parallel performance. Weak scaling studies analyze the performance with a fixed problem size per node. Therefore, weak scaling studies deal with problems of variable size.

Since we expect transport sweeps to dominate run time of an MOC solver, such as OpenMOC, this thesis focuses on just the scalability of transport sweeps. However, it is important to note that domain decomposing the CMFD acceleration is critical to being able to accelerate using any significant number of CMFD groups since a many-group CMFD can be quite costly for a large problem when solved on a single node. However, once the problem is distributed over the available nodes, its cost becomes trivial in comparison with the transport sweeps.

In each trial, only one MOC transport sweep is conducted. Each domain decomposed configuration is simulated three times. Each node is assigned a domain and each node makes use of all of its cores with shared memory parallelism, as outlined in previous chapters. The presented results take into account the average run time, as well as the minimum and maximum runtime to form estimates of the uncertainty. All presented results in this chapter use the Argonne BlueGene/Q supercomputer on the Cetus partition. The Argonne BlueGene/Q supercomputer is designed for many-node parallelism, reserving a large number of adjacent nodes for each submitted job. The reservation of many adjacent nodes allows for much better reproducibility in timing studies. The results in this chapter focus on solving geometries found in the BEAVRS benchmark. These geometries may be replicated in a lattice for weak scaling studies.

6.5.1 Strong Scaling Studies

For the strong scaling studies the single assembly modeled detailed in Appendix E.2.3 is chosen. It is important to note that this model includes full axial detail, including grid spacers and axial water reflectors. The problem is then domain decomposed only in the axial direction.

The MOC ray spacing parameters for these strong scaling tests are given in Table 6.1. Due to the one hour time limit of jobs on the Cetus partition of the Argonne BlueGene/Q supercomputer, these parameters are significantly coarser than those required to accurately converge the fission source distribution. When domain decomposing the geometry, tracks are generated such that the track laydown is guaranteed to be the same for all the domain decomposed configurations.

The strong scaling results are presented in Figure 6-8 for both linear source and flat source solvers. For these results the uncertainties are not shown because they are so small in comparison with differences in run-time from scaling, causing them to not be noticeable.

Note that the scaling is far better for the linear source solver than the flat source solver since there is significantly more on-node computational work for the linear source

Table 6.1: MOC parameters for the strong scaling studies of the single assembly test problem

Number of Sectors in Moderator	8
Number of Sectors in Fuel	4
Height of Flat Source Regions	2.0 cm
Radial Ray Spacing	0.1 cm
Axial Ray Spacing	1.5 cm
Number of Azimuthal Angles	16
Number of Polar Angles	6

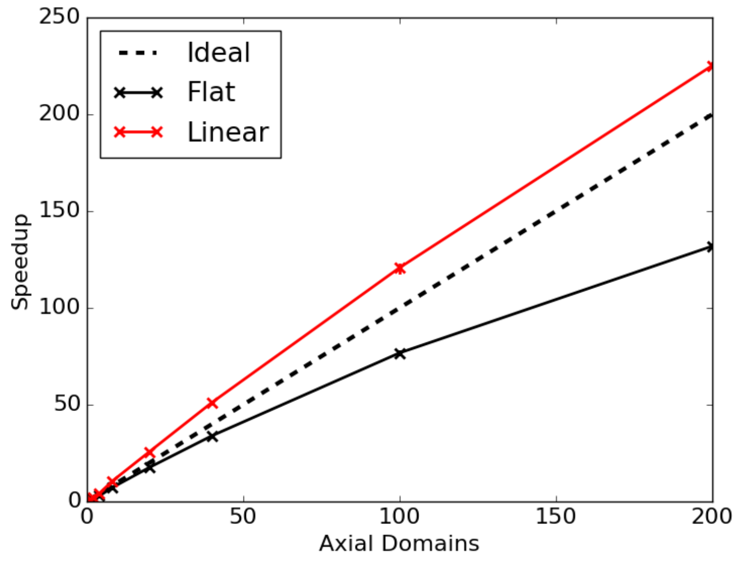


Figure 6-8: Strong scaling of axial domain decomposition for a single assembly model with the flat and linear source solvers.

solver while the inter-node communication costs are the same as the flat source solver. In fact, the strong scaling performance for the linear source solver is better than the ideal linear scaling. Since the aim of this thesis is to solve full core problems with the linear source solver, its performance will be the main focus of discussion.

These results might be difficult to understand as performance would not be expected to exceed ideal. However, recall from Chapter 5 that additional intersections, and therefore segments, are inserted when a piece-wise axially extruded geometry is converted to an axial extruded geometry on each sub-domain. As a given geometry is domain decomposed axially, the deviations from a true axially extruded geometry on each sub-domain decrease, and therefore the number of additional intersections inserted to ensure an axially extruded geometry on each sub-domain also decrease.

This implies that runtime should be normalized by the number of segments to capture the pure computational performance of the algorithm. The results when normalizing runtime by the number of segments is shown in Figure 6-9.

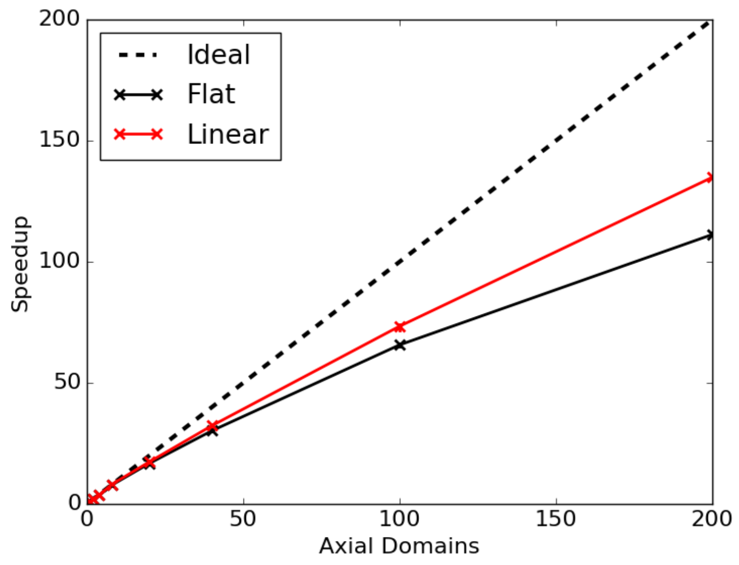


Figure 6-9: Strong scaling of axial domain decomposition for a single assembly model of the (a) flat and (b) linear source solver, normalized by the number of segments treated in the simulation.

After the normalization, the linear source results are no longer better than ideal. It is important to note that here a fixed 400 cm tall single assembly is domain decomposed axially all the way to 200 domains (each of 2 cm height), which is quite excessive. At 200 domains there is only one source region axially for each domain. The expected MOC ray parameters to fully resolve the fission distribution produce approximately 27× more rays than the coarse ray parameters simulated here, greatly increasing the on-node work and improving the domain decomposition performance. Using finer MOC ray parameters on this benchmark, we would expect to only domain decompose to 20 domains axially. For a problem of this size with the ray parameters given in Table 6.1, any domain decomposition might be excessive. Still at 8 axial domains, the speedup of the linear source solver is 97% of ideal. The efficiency then decreases to 87% and 67%

at 20 and 200 axial domains, respectively.

Since the desired MOC parameters are significantly finer than those shown in these scaling studies, the performance should be tested using the desired parameters. Due to the one hour time limit on Cetus runs, this can only be tested using at least several domains. Therefore, the linear source solve is tested with a domain decomposition of 20 axial domains and with the the expected MOC parameters to accurately converge the fission distribution, given in Table 6.2.

Table 6.2: Expected MOC parameters to accurately converge a full core PWR fission distribution

Number of Sectors in Moderator	8
Number of Sectors in Fuel	4
Height of Flat Source Regions	2.0 cm
Radial Ray Spacing	0.05 cm
Axial Ray Spacing	0.75 cm
Number of Azimuthal Angles	64
Number of Polar Angles	10

From the strong scaling studies with coarse rays, 67% efficiency was observed for the linear source solver with 20 axial domains. In order to judge the efficiency with desired ray parameters, the timing breakdown for the single assembly case is presented in Table 6.3. Here, three timing statistics are considered: total transport sweep execution time, angular flux communication time, and idle time between sweeps. The angular flux communication time is the total time required to communicate all angular flux information between domains. The idle time between sweeps is the average time that nodes remain idle after doing all on-node work and before transferring angular fluxes with other domains. Therefore, the idle time is an indication of load imbalance.

These results show the angular flux communication time to be almost trivial for this single assembly case but the idle time is measurable. This indicates a cost to the modular ray tracing structure where domains are required to be of equal size. For this single assembly problem, there are many more segments in the core, and thus more work, than in reflector regions. Domains in the reflector region need to wait for in-core domains to

Table 6.3: Timing breakdown of the single assembly test problem with the linear source solver domain decomposed into 20 axial domains

Procedure	Time (s)	% of Total Transport Sweep
Total Transport Sweep	1059 +/- 11	–
Angular Flux Communication	15 +/- 8 $\times 10^{-3}$	1.4
Idle Time Between Sweeps	145 +/- 10	13.7

finish their work before moving on, leading to wasted computational resources. Still, the combined 15.1% of transport sweep time spent to accommodate domain decomposition is a small cost for a problem of this size. Since this problem is run with the desired MOC parameters, it should be a good measure of the expected performance on realistic problems.

6.5.2 Weak Scaling Studies

For the weak scaling studies, chosen geometries are replicated in a lattice. The first set of tests radially replicate the single assembly geometry used in the previous section and detailed in Appendix E.2.3 in a 2D lattice. These series of tests are true to the design of common reactor cores, as they typically resemble arrangements of assemblies placed in some radial format. The disadvantage of this test problem is that it only tests weak scaling of the domain decomposition in the radial directions. Therefore, another set of tests is performed that replicate the SDSA test problem detailed in Appendix E.2.5 in a 3D lattice. This test problem more precisely tests the scalability of the algorithm as every domain has the same geometry and material composition.

Since the emphasis of this work is to both solve real problems and solve them efficiently, these tests provide great insight into the capabilities and behavior of the OpenMOC implementation. For these weak scaling tests, all test problems use the desired MOC parameters, as given in Table 6.2. All results in this section focus on the scalability of the linear source solver.

6.5.2.1 2D Lattice of the Single Assembly Geometry

First, the 2D lattice of full-height assemblies is considered. The geometry needs to be domain decomposed axially in order to run with these desired parameters in less than the one hour time limit set by the Cetus policies. Therefore, all geometries are domain decomposed into 20 axial domains and then further domain decomposed such that each domain simulates an assembly in the radial direction. The base case is therefore the single assembly decomposed into 20 domains, which was analyzed at the end of the strong scaling studies.

All efficiency results are relative ot the base case of a single assembly. The ideal case is for runtime to stay fixed as the problem size is increased with 20 nodes assigned to each assembly. The geometry is expanded in a $N \times N$ square lattice where N is the number of domains in both the x and y directions. The results are shown in Fig. 6-10 where the efficiency is plotted as a function of the number of nodes used in the computation. Note the logarithmic scale on the x-axis.

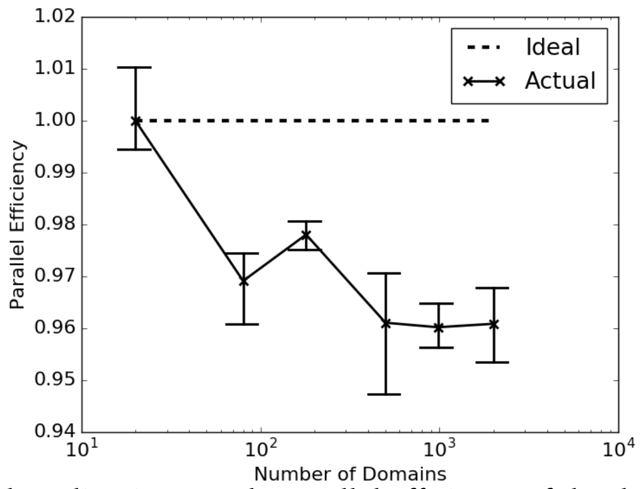


Figure 6-10: Weak scaling inter-node parallel efficiency of the domain decomposition implementation of the linear source solver on a replicated 2D lattice of assemblies.

These results show the domain decomposition implementation is able to efficiently scale to many nodes. Even at 2000 nodes (32,000 cores), the efficiency is above 95%.

6.5.2.2 3D Lattice of the SDSA Geometry

The next sets of tests are similar to the 2D lattice tests but expand the domain in all 3 Cartesian directions using a 3D lattice. Instead of replicating the single assembly test problem, the SDSA test problem is replicated, causing each domain to have the same geometry and equal computational work. The geometry is replicated in a $N \times N \times N$ cubic lattice where N is the number of domains in each Cartesian direction.

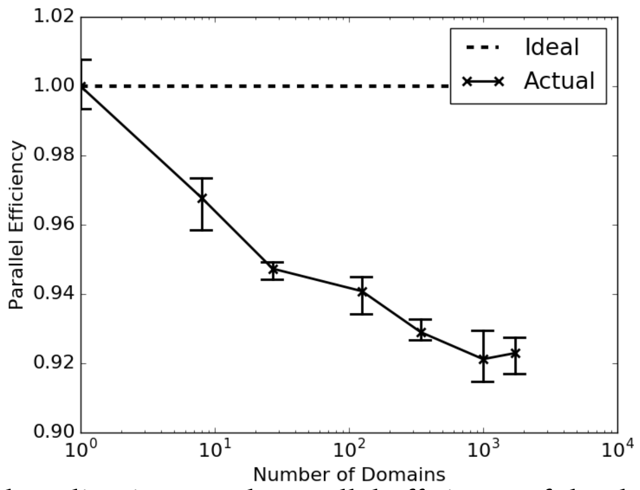


Figure 6-11: Weak scaling inter-node parallel efficiency of the domain decomposition implementation of the linear source solver on a replicated 3D lattice of the SDSA test problem.

Again, excellent scaling is observed with the efficiency above 90% for all tested cases. At 1728 nodes (27,648 cores) the efficiency is 92%.

6.6 Conclusion

In order to scale to large problems, inter-node parallelism is essential. To accomplish this, domain decomposition has been implemented in OpenMOC with MPI, utilizing the natural track linking of the MRT structure. This allows angular fluxes to be easily communicated by referring to connecting periodic tracks. The domain decomposition implementation has been tested and observed to scale very well to many nodes. Communication costs of transferring angular fluxes were at most only a few percent of overall runtime for realistic cases. The biggest hindrance to inter-node scaling of the domain decomposition implementation is the potential load imbalance created by domains of unequal work. For domains with uniform work, the inter-node parallel efficiency was above 90% for all weak scaling studies. Significant degradation in performance was only noticed in strong scaling studies for cases with an unrealistically low amount of on-node work.

Highlights

- Domain decomposition is implemented by partitioning the geometry into regions of equal dimensions with tracks naturally linking at sub-domain boundaries due to the MRT ray tracing scheme
- Both MOC and CMFD solvers are domain decomposed with MPI used to communicate information between nodes, often using the Buffered Synchronous Algorithm
- Currents at corner and edge boundaries need to be carefully treated in a domain decomposed setting to ensure consistency between MOC and CMFD solvers
- The largest hindrance to the scalability of the domain decomposition implementation is the potential for load imbalance between sub-domains of equal size but unequal work
- Results show the domain decomposed implementation to have very low communication costs, allowing for over 90% parallel scaling in weak scaling studies with expected MOC parameters

Chapter 7

Convergence of MOC Source Iteration

In the previous chapters, the implementation of methods in OpenMOC has been discussed in great detail. All of these methods rely on the MOC form of the transport equation, presented in Chapter 2. More specifically, these methods rely on source iteration to converge the solution. In this chapter, the MOC equations are addressed from a general linear algebra perspective illuminating the deficiencies of source iteration, whose stability is addressed. While convergence is straightforward for physical cross-sections, transport correction can cause convergence issues. A stabilization technique is proposed, termed diagonal stabilization, which can alleviate the convergence issues.

7.1 Introduction

The MOC system of equations leads to a linear system of the form

$$\boldsymbol{\phi} = J\left(\frac{1}{k}F + S\right)\boldsymbol{\phi}. \tag{7.1}$$

where $\boldsymbol{\phi}$ is a vector representing all scalar fluxes and J, F, and S are matrices. Matrix-vector multiplications with J are termed transport sweeps in which the MOC equations are applied over segments. A more complete description of the MOC linear system is found in Appendix A in which the structure of matrices J, F, and S are explicitly identified. The form in Eq 7.1 is not unique to MOC, but rather many methods solve the multi-group transport equation in the same form, differing only in the way matrix-vector products with J are computed.

For physical cross-sections, the matrices J, F, and S are all positive. However, when transport correction is introduced, discussed further in Appendix F.3, components of the scattering matrix can become negative. Specifically, for a region i and energy group g, the transport correction $\Delta\Sigma_{tr}^{i,g}$ is applied to both the total cross-section $\Sigma_t^{i,g}$ and within-group scattering cross-section $\Sigma_s^{i,g\to g}$ resulting in a transport cross-section $\Sigma_{tr}^{i,g}$ and transport-corrected within-group scattering cross-section $\tilde{\Sigma}_{s,i}^{g\to g}$ as

$$\begin{aligned}\Sigma_{tr}^{i,g} &= \Sigma_t^{i,g} - \Delta\Sigma_{tr}^{i,g} \\ \tilde{\Sigma}_s^{i,g\to g} &= \Sigma_s^{i,g\to g} - \Delta\Sigma_{tr}^{i,g}\end{aligned} \tag{7.2}$$

When using large numbers of energy groups, the within-group scattering cross-sections can become small, and the modified within-group scattering cross-section can become negative with transport correction. Tabuchi discovered negative within-group scattering cross-sections to be an issue when converging within-group scattering iterations of MOC [59]. The study used an iteration scheme in which the spatial distribution is converged for each estimate of the neutron source. The iterations to converge the spatial distribution are termed *inner iterations*. It was identified that the iteration matrix for inner iterations could have a spectral radius greater than unity, causing the system not to converge. Tabuchi later proposed a stabilization technique for inner iterations [60].

7.2 Equivalence with Collision Probability Methods

Deterministic methods such as flat source MOC are equivalent to the collision probability form, aside from discretization errors [59]. Although collision probabilities do not explicitly enter the MOC equations, the neutron balance equation takes the form in Eq. 7.3:

$$\Sigma_{tr}^{i,g}\phi_{i,g}V_i = \sum_j P_{ji,g}q_{j,g}V_j \tag{7.3}$$

where $P_{ji,g}$ represents the collision probability of a neutron of energy group g from region j to region i, V_i represents the volume of region i, $q_{i,g}$ represents the neutron source and $\phi_{i,g}$ represents the average scalar flux in region i and group g. In general, the neutron source can be computed by summing contributions from all G energy groups as

$$q_{i,g} = \sum_{g'=1}^{G} \left(\Sigma_s^{i,g'\to g} \phi_{i,g'} + \chi_{i,g} \nu\Sigma_f^{i,g'} \phi_{i,g'} \right) \tag{7.4}$$

where $\Sigma_s^{i,g'\to g}$ is the scattering cross-section in region i from group g' to group g, $\chi_{i,g}$ is the fission emission probability for group g in region i, and $\nu\Sigma_f^{i,g'}$ is the fission production in region i from group g'. Combining this definition with Eq. 7.3, as well as the reciprocity relationship,

$$P_{ij,g} \Sigma_{tr}^{i,g} V_i = P_{ji,g} \Sigma_{tr}^{j,g} V_j, \tag{7.5}$$

neutron balance can be presented in the form of Eq. 7.6.

$$\phi_{i,g} = \sum_j \frac{P_{ij,g} \sum_{g'=1}^{G} \left(\Sigma_s^{j,g'\to g} \phi_{j,g'} + \chi_{j,g} \nu\Sigma_f^{j,g'} \phi_{j,g'} \right)}{\Sigma_{tr}^{j,g}} \tag{7.6}$$

Notice that this is in the form of Eq. 7.1. Therefore, the matrix $A = J\left(\frac{1}{k}F + S\right)$, with rows and columns indexed by (i,g) where i is the region and g is the energy group, can be expressed as

$$A_{(i,g),(j,g')} = P_{ij}^{g} \left(\frac{\chi_{j,g} \nu\Sigma_f^{j,g'}/k + \Sigma_s^{j,g'\to g}}{\Sigma_{tr}^{j,g}} \right). \tag{7.7}$$

Matrix A is square and dense with length equal to the number of scalar fluxes.

7.3 Iteration Schemes

7.3.1 Power Method

The power method is one common method to solve an eigenvalue problem, such as the form given in Eq. 7.1, which yields the dominant eigenvector corresponding to the steady-state flux distribution. To invoke the power method, Eq. 7.1 can be re-arranged to the form in Eq. 7.8 where I represents the identity matrix.

$$\begin{aligned} Z &\equiv (I - JS)^{-1} JF \\ Z\boldsymbol{\phi} &= k\boldsymbol{\phi} \end{aligned} \tag{7.8}$$

With the power method scheme, repeated multiplication by Z yields the dominant eigenvector [61]. Rather than performing strict power method iterations, the estimate of the eigenvalue k_n at iteration n is often included in the source. This scheme is presented in Eq. 7.9. These iterations are often termed *outer iterations*.

$$\boldsymbol{\phi}_{n+1} = (I - JS)^{-1} J \frac{F}{k_n} \boldsymbol{\phi}_n \tag{7.9}$$

Since explicitly taking the matrix inverse of $I - JS$ is unwise, an iterative scheme is necessary to solve the linear system. In many transport methods, such as MOC, computing each element of J can be just as expensive as computing a matrix-vector product. In addition, each matrix-vector product with matrix J is usually quite expensive (often termed *transport sweeps*). Therefore, few methods are available to efficiently solve the linear system in practice.

7.3.2 Source Iteration

An alternative way to solve the transport equation is to directly apply the transport sweep to the full neutron source. Note that the neutron source $\mathbf{q}$ can be computed as

$$\mathbf{q} = \left(\frac{1}{k} F + S \right) \boldsymbol{\phi}.$$

The transport sweep matrix J therefore yields the scalar flux distribution associated with the computed source distribution. In this form, a new iterative process could be introduced in which a new source distribution is computed at each iteration, yielding a new estimate of the associated scalar flux distribution. Therefore, solving the transport equation in this form where source terms are lagged is termed *source iteration*. Specifically, the eigenvalue problem in Eq. 7.1 is iteratively solved with the left hand side updated and the right hand side lagged as

$$\boldsymbol{\phi}_{n+1} = J\left(\frac{F\boldsymbol{\phi}_n}{k_n} + S\boldsymbol{\phi}_n\right). \tag{7.10}$$

It is important to note that this process is non-linear due to the iteration matrix $J\left(\frac{1}{k_n}F + S\right)$ being dependent on the iteration number n. To simplify this relationship, assume that the eigenvalue k is perfectly known to be k_{crit}, the eigenvalue associated with the dominant mode of the system. In reality, the exact value of k_{crit} is not known a priori but observations and intuition suggests that it does not strongly impact convergence. With the eigenvalue fixed, the system becomes

$$\boldsymbol{\phi}_{n+1} = A\boldsymbol{\phi}_n \tag{7.11}$$

where matrix A is defined in Eq. 7.7 with $k = k_{crit}$. This process is equivalent to power method iterations with the matrix A, which converges to the eigenvector associated with the dominant eigenvalue of A. Since k_{crit} is the dominant eigenvalue of the original system, 1.0 must be an eigenvalue of A. In addition, if an everywhere positive solution exists, it must be associated with the physical solution.

Recall from Eq. 7.7 that the iteration matrix A is everywhere positive and real as long as the within-group scattering cross-section is positive. According to the Perron-Frobenius Theorem [62], square matrices of all positive real entries have a unique largest real eigenvalue which is dominant and associated with an everywhere positive eigenvector. In addition, all other eigenvectors must have a negative component. Without transport correction, the iteration matrix A is an all positive and real matrix, implying that the largest eigenvalue is 1.0 and is associated with the physical solution. Therefore, the process detailed in Eq. 7.10 will converge to the physical solution.

However, with the transport correction, it is possible to have negative within-group scattering, causing this condition not to hold. Therefore, the system might still converge, but convergence cannot be guaranteed under the Perron-Frobenius Theorem. Therefore, the iteration scheme should be updated to ensure convergence.

7.4 Stabilization of Source Iteration

Note that Eq. 7.12 is a mathematically valid rewriting of Eq. 7.1 for *any matrix D* where $I + D$ is invertible.

$$\boldsymbol{\phi} = (I + D)^{-1}\left[J\left(\frac{1}{k}F + S\right) + D\right]\boldsymbol{\phi} \tag{7.12}$$

Next, the same source iteration scheme is applied where the left hand side is updated with the right hand side constant. Since $I + D$ needs to be easily invertible for this new scheme to be efficient, D is chosen to be diagonal. The convergence discussion then follows the same discussion as before except the new iteration matrix now has the form $\tilde{A} = (I + D)^{-1}\left[J\left(\frac{1}{k_{crit}}F + S\right) + D\right]$ with the matrix $\tilde{A}$ is described by

$$\tilde{A}_{(i,g),(j,g')} = \frac{1}{1 + D_{(i,g),(i,g)}}\left[P_{ij}^{g}\left(\frac{\chi_{j,g}\,\nu\Sigma_f^{j,g'}/k_{crit} + \Sigma_s^{j,g'\to g}}{\Sigma_{tr}^{j,g}}\right) + D_{(i,g),(j,g')}\right] \tag{7.13}$$

where the rows and columns are again indexed by region, energy group pairs. The diagonal elements of D are chosen to be:

$$D_{(i,g),(i,g)} = \begin{cases} \frac{-\rho\Sigma_s^{i,g\to g}}{\Sigma_{tr}^{i,g}}, & \text{for } \Sigma_s^{i,g\to g} < 0 \\ 0, & \text{otherwise} \end{cases} \tag{7.14}$$

where the damping coefficient ρ is a positive value chosen by the user. For $\rho = 1$, the diagonal update ensures that there are no negative diagonal elements in the iteration matrix.

The diagonal stabilization scheme has the effect of shifting the iteration matrix eigenvalues to be more positive. The intuition for this shift is due to the Gershgorin Disk Theorem where eigenvalues exist in disks around diagonal elements. Since the diagonal elements are increased and all matrix elements are contracted, the Gershgorin Disks are likewise shifted positive with their radii contracted. While this improves stability, it also tightens the positive-eigenvalue modes, causing larger dominance ratios in the iteration matrix, and slower convergence. Therefore, ρ should be chosen to be small while still ensuring convergence. An illustration of the shift in eigenvalues is shown in Figure 7-1.

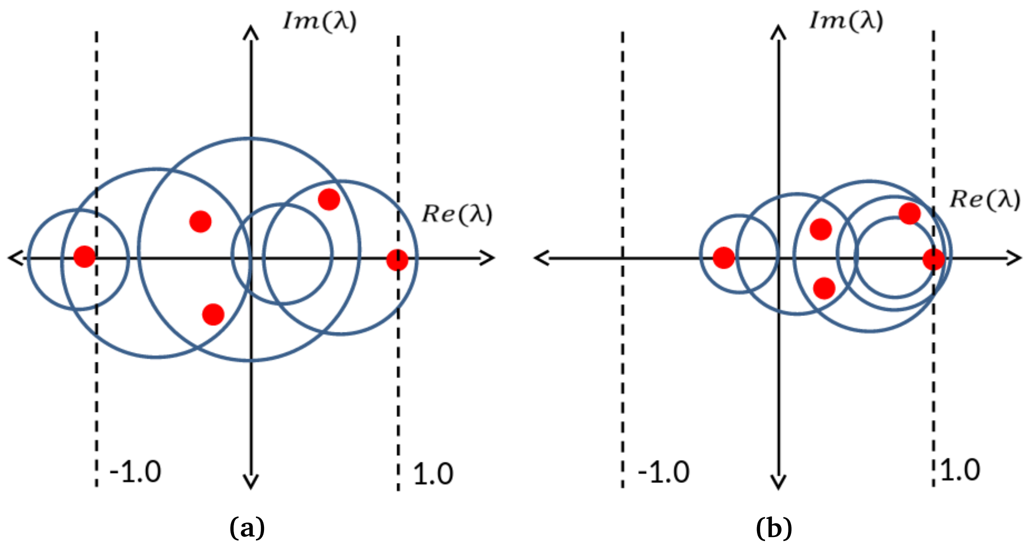


Figure 7-1: The effect of diagonal dominance shifting the eigenvalues λ of a matrix with initial Gershgorin disks (blue) and eigenvalues (red) in the complex plane (a) shifted positively with diagonal stabilization (b). Note that the eigenvalue at 1.0 stays at 1.0.

Tabuchi suggested a similar scheme for converging inner within-group scattering iterations of MOC transport sweeps [60]. Though originally limited to MOC transport sweeps with inner within-group scattering iterations, Tabuchi's scheme is equivalent to the diagonal stabilization scheme presented here for damping the scalar flux update except for the choice of diagonal matrix $\tilde{D}$ in place of D with elements

$$\tilde{D}_{(i,g),(i,g)} = \max_{j} \left| \frac{\Sigma_s^{j,g\to g}}{\Sigma_{tr}^{j,g}} \right| \tag{7.15}$$

which is far larger than the equivalent formulation given in Eq. 7.14. Not only is the damping applied to all fluxes, not just those with negative within-group scattering, but it also takes the maximum ratio of within-group scattering to transport cross-section $\Sigma_s^{j,g\to g}/\Sigma_{tr}^{j,g}$ across all regions j, which may already be positive.

It is important to note that in the OpenMOC implementation, negative sources are set to zero in the first 20 iterations. In addition, negative fluxes are always set to zero.

Whenever the correction is applied during an iteration, a warning message is displayed, alerting the user to the behavior. Negative sources and fluxes are an important indicator of convergence issues, as a solution corresponding with a negative eigenvalue will have negative components in its associated eigenvector. While setting negative sources and fluxes to zero helps the convergence behavior by more quickly eliminating non-physical behavior, it does not remedy the fundamental convergence issues introduced by negative cross-sections without the use of a stabilization technique such as diagonal stabilization.

7.5 Convergence Criteria

Since the MOC equations are converged iteratively with source iteration, a convergence tolerance must be chosen for the simulation. In this thesis, convergence is determined by iterative change in the neutron production rate. Specifically, the neutron production rate f_i in a source region i can be computed with a sum of contributions from all G energy groups as

$$f_i = \sum_{g=1}^{G} \nu\Sigma_f^{i,g} \overline{\phi_{i,g}} \tag{7.16}$$

for the current estimate of scalar fluxes $\overline{\phi_{i,g}}$. The relative change in fission rate, termed `eps-MOC`, in iteration n can be computed as

$$\texttt{esp-MOC} = \frac{1}{N_f} \sqrt{\sum_{i \in N_f} \left(\frac{f_i^n - f_i^{n-1}}{f_i^{n-1}} \right)^2} \tag{7.17}$$

where N_f is the number of regions with a nonzero neutron production rate and f_i^n refers to the neutron production rate of region i in iteration n. For all results in this thesis, convergence is determined when `esp-MOC` is reduced to 10^{-4} and the change in eigenvalue estimate from the previous iteration is less than 1 pcm.

7.6 Convergence of Source Iteration

In order to test the convergence behavior of the stabilization methods, the stabilization methods were implemented in OpenMOC. The single assembly test problem described in Appendix E.2.3 is used to test convergence behavior. Results in this section focus on the flat source 3D MOC solver in OpenMOC for simplicity, using the MOC parameters given in Table 7.1. While finer MOC parameters would be necessary to accurately resolve the solution, convergence has not been observed to change substantially from refining MOC parameters.

Table 7.1: MOC parameters for single assembly convergence studies

Source Approximation	Flat
Number of Sectors in Moderator	8
Number of Sectors in Fuel	4
Height of Flat Source Regions	2.0 cm
Radial Ray Spacing	0.1 cm
Axial Ray Spacing	1.5 cm
Number of Azimuthal Angles	32
Number of Polar Angles	6
Domain Decomposition	$1 \times 1 \times 100$

To analyze the convergence behavior, an error metric is needed. Usually, error is analyzed in terms of Root Mean Square (RMS) error in the fission distribution over all fissile regions. During simulations, this can only be compared in terms of results from previous iterations. However, for these convergence studies, a reference solution is used which represents the final fission distribution of a tightly converged case. Therefore, error can reliably be assessed rather than the difference with previous iterations.

In order to focus on the asymptotic convergence behavior, rather than the behavior far from convergence, the MOC simulations start with a *good guess* of the scalar flux distribution. Specifically, the MOC convergence criteria is chosen to be extremely tight and a converging simulation where the error drops below 3×10^{-5} on RMS fission rate error is chosen as the starting point.

The convergence behavior is presented in Figure 7-2 in which the relative error is plotted as a function of iteration number for a variety of stabilizing schemes. Notice

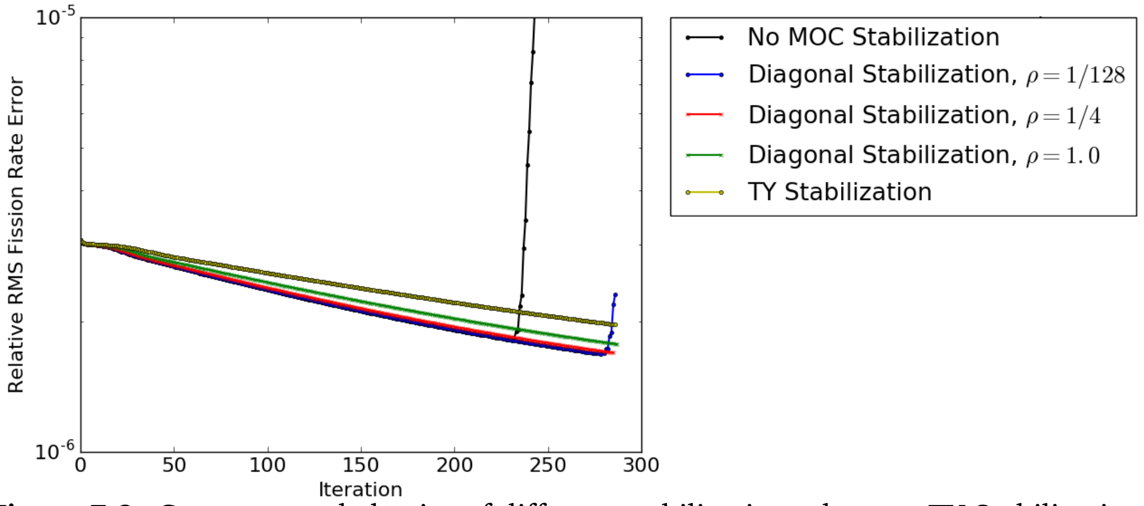


Figure 7-2: Convergence behavior of different stabilization schemes. TY Stabilization refers to the approach presented by Tabuchi whereas Diagonal Stabilization refers to the approach presented in this paper with damping coefficient ρ.

that the case without stabilization diverges while all the stabilization cases converge except for the new diagonal stabilization scheme with damping factor $\rho = 1/128$. This is expected since as ρ approaches zero, the stabilization method becomes equivalent to source iteration without stabilization. This clearly shows that this problem requires stabilization to converge with source iteration. In addition, the more conservative the stabilization scheme, the more convergence is slowed. The scheme proposed by Tabuchi slows convergence the most. The diagonal damping scheme hinders convergence significantly less, especially for $\rho < 1$.

7.7 Convergence Results with CMFD Acceleration

In Section 7.6, the convergence of source iteration and its stability properties were discussed. When CMFD acceleration is applied, as detailed in Appendix B, the behavior is much more difficult to analyze since CMFD is a non-linear process. Therefore, discussion of CMFD results will focus on hypotheses to explain the observed results rather than rigorous analysis.

In this section, the CMFD mesh is always chosen to be uniform with cells of pin-cell pitch in the radial dimensions and 2.0 cm in the axial dimension. In addition, a relaxation factor of 0.7 is applied to the CMFD acceleration scheme for computing corrected diffusion coefficients. Many CMFD implementations choose to collapse to a fewer number of energy groups in order to improve the speed of the CMFD solver. While the CMFD equations are computationally less expensive in collapsed few-group structures, it is possible that collapsing group structures could cause slower convergence by not fully resolving spectral effects. Results with a variety of CMFD group structures are presented. While the MOC calculation always uses a 70 group structure, CMFD group structures are formed with G_C CMFD groups, with group ranges taken from the CASMO-4 Manual [63], as presented in Appendix C.

All of the stability results presented in this section use the parameters in Section 7.1 unless otherwise specified.

7.7.1 Single Assembly Convergence with Water Reflectors

The single assembly model previously analyzed for pure source iteration in Section 7.6 is first analyzed with CMFD acceleration using the same parameters. It is important to remember that this problem includes axial water reflectors. These large regions of water where the transport-correction is large could potentially have a strong impact on convergence. The results with CMFD applied and *without* any MOC source iteration stabilization are shown in Figure 7-3 for a variety of CMFD group structures with the error once again relative to a tightly converged reference. As expected, most of the trials have trouble converging. This is likely due to the convergence issue observed with pure source iteration. However, the 70 group CMFD scheme appears to be stable. It is important to remember that the CMFD equations do not suffer from the issue of a negative dominant eigenvalue since the CMFD equations can be written such that within-group scattering does not arise when removal cross-sections are used instead of total cross-sections.

If 70 group CMFD is driving the convergence process, then the instability in the underlying source iteration technique appears to be overcome by the CMFD acceleration

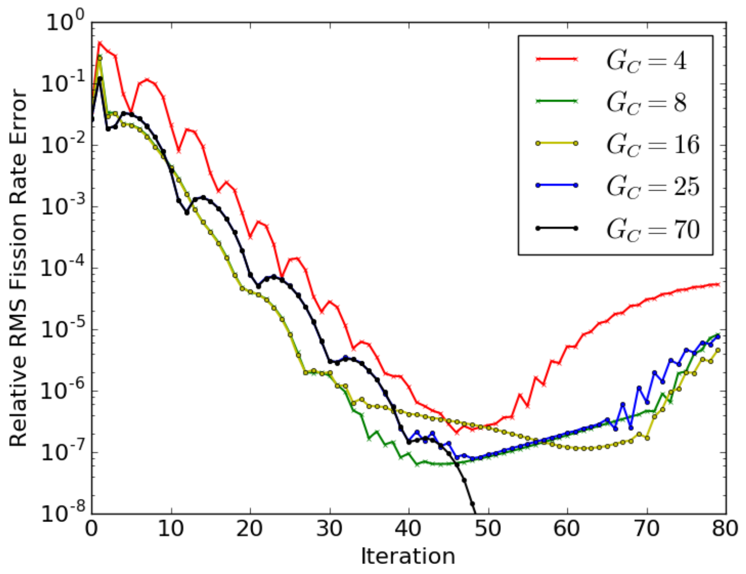


Figure 7-3: Convergence behavior for a variety of CMFD group structures with G_C CMFD groups and *without* MOC source iteration stabilization.

prolongation in which MOC scalar fluxes are updated with the CMFD solution. One way of interpreting this behavior is to think of the convergence as being dominated by the CMFD acceleration where MOC transport sweeps are used purely to resolve local behavior, integrate reaction rates to form CMFD cross-sections, and tally currents crossing CMFD mesh surfaces.

However, with a reduced CMFD group structure, the flux distribution in energy within a single CMFD group *must* be resolved by the MOC source iteration process. Since the underlying MOC source iteration process is unstable, any reliance on the process for convergence using a reduced group structure appears to lead to instability.

7.7.2 Single Assembly Convergence with Stabilization

To verify that the convergence issue can be remedied with MOC source iteration stabilization, the same single assembly test is conducted but using diagonal stabilization with $\rho = 1/4$. For pure source iteration, this was sufficient for stabilization, as previously shown in Figure 7-2. The results using this stabilization with CMFD acceleration are shown in Figure 7-4. As expected, the MOC source iteration stabilization fixes the convergence issues for all CMFD group structures, indicating that the convergence issue

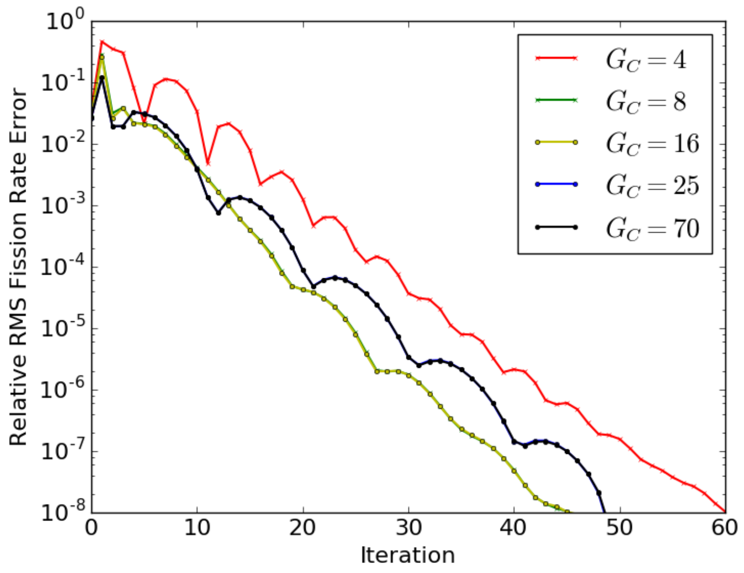


Figure 7-4: Convergence behavior for a variety of CMFD group structures with G_C CMFD groups and *with* Diagonal MOC source iteration stabilization ($\rho = 1/4$).

was caused by the underlying MOC source iteration.

7.7.3 Single Assembly without Axial Water Reflectors

Next, the truncated single assembly model is studied, as detailed in Appendix E.2.4. This model is the same as the single assembly model but without the axial water reflectors, using a $1 \times 1 \times 90$ domain decomposition. Without reflectors, this model is more similar to lattice physics models. The convergence behavior is analyzed *without* source iteration stabilization. The results with CMFD acceleration for a variety of CMFD group structures are presented in Figure 7-5. Notice that the convergence issues are not present in this model. This indicates the issue is caused by the axial water reflectors. More generally, it appears to be a problem with deep water reflectors that have negative cross-sections. Recall from the theory discussion that iteration matrix components in Eq. 7.7 were formed with the product of collision probability and self-scattering cross-sections. In deep water reflectors, the probability of transport between water regions containing negative cross-sections is significantly higher than in the active fuel where many neutrons collide with fuel. This motivates neutron behavior in deep reflectors causing the instability.

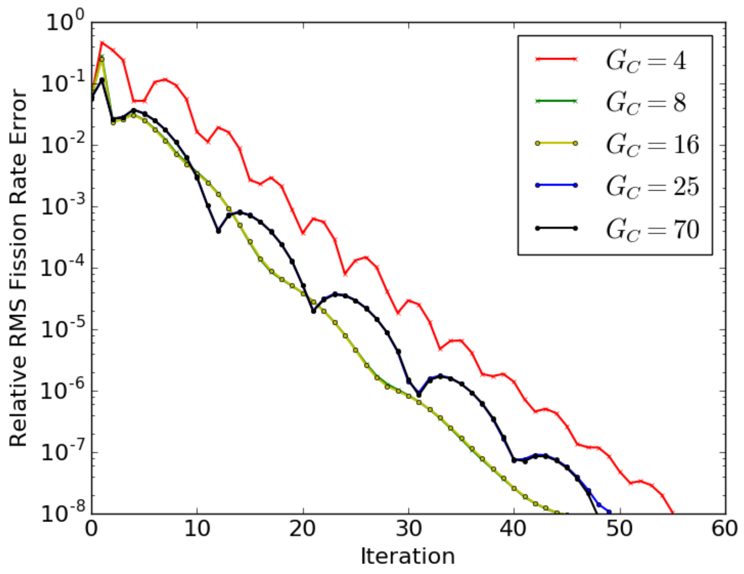


Figure 7-5: Convergence behavior for the single assembly *without* axial reflectors with a variety of CMFD group structures with G_C CMFD groups and *without* MOC source iteration stabilization.

7.7.4 Full Core Behavior

The previous results were focused on a single assembly with and without axial reflectors which proved the existence of source iteration instability. Since this issue seems to only appear at very low errors, the reader might judge this issue to be purely academic. However, for problems where deeper reflectors exist, such as a typical full core LWR problem, this issue is exacerbated. To test this effect, full core models are simulated using the same cross-section set applied in the single assembly studies. The geometry outside the core is radially discretized into a 3×3 square grid within each pin-cell-sized region.

7.7.4.1 2D Extruded Model

Before simulating the explicit 3D model, the 2D extruded model detailed in Appendix E.2.2 is studied. This model lacks the axial water reflectors but still contains significant water reflectors in the radial direction. A $17 \times 17 \times 1$ domain decomposition is used. The results are shown in Figure 7-6. The simulation is run with 8 group and 70 group CMFD with Diagonal Stabilization. The damping coefficient ρ is chosen for both cases to be

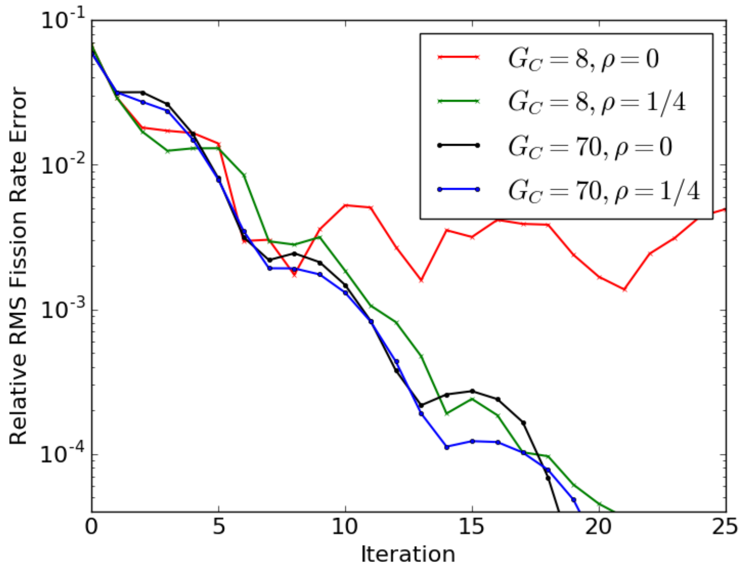


Figure 7-6: Convergence behavior of CMFD group structures with G_C CMFD groups and Diagonal MOC source iteration stabilization with stabilization coefficient ρ.

zero (equivalent to no stabilization) and 1/4. For all reduced CMFD group structures, similar results are observed as seen with the 8 group CMFD structure in Figure 7-6 in which reasonable convergence can only be obtained by applying the MOC diagonal stabilization.

7.7.4.2 Explicit 3D Model

Now the full core 3D BEAVRS model is simulated, as detailed in Appendix E.2.1. Since the problem is large, the MOC angular quadrature is significantly coarsened to 4 azimuthal angles and 2 polar angles in order to run a significant number of iterations on the Cetus partition of the Argonne BlueGene/Q supercomputer. These parameters are very coarse, but still exhibit the convergence issue. Since the run time with coarse angles can become dominated by the CMFD solution time, it is infeasible to use 70 CMFD groups on the Cetus partition. Therefore, results are presented only for a reduced 8 group CMFD structure. A $17 \times 17 \times 5$ domain decomposition is used. The results are shown in Figure 7-7 both with and without using the Diagonal Stabilization technique with $\rho = 1/4$.

These results show that for the 3D full core PWR problem, a stabilization tech-

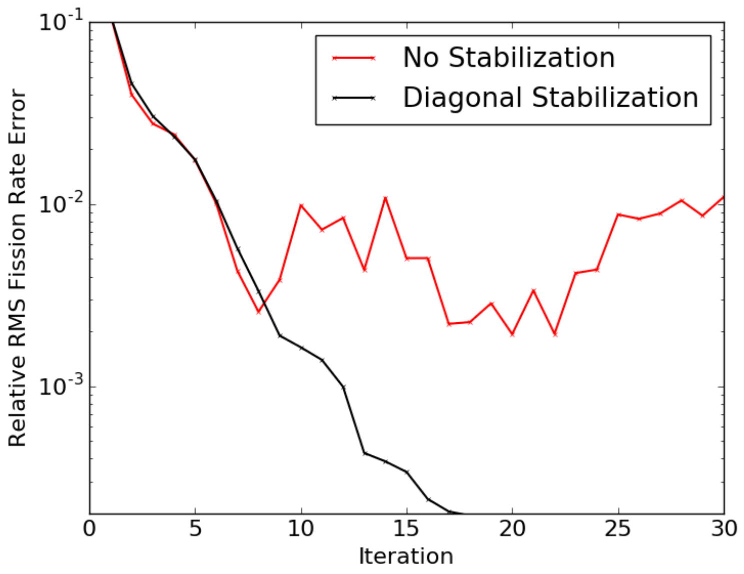


Figure 7-7: Convergence behavior of OpenMOC on the full core BEAVRS benchmark with and without Diagonal Stabilization ($\rho = 1/4$).

nique is necessary in order to reach any reasonable convergence, especially when it is computationally infeasible to use a many-group CMFD solver.

7.7.4.3 Explicit 3D Model with Linear Source

The preceding tests have all been conducted using a flat source approximation. However, the goal of this thesis is to solve full core LWR problems with a linear source approximation. While the theory of the diagonal stabilization was developed for flat source, it can also be applied to simulations with a linear source approximation. In the OpenMOC implementation, the same damping of each flat source flux is applied to the associated linear source moments. Again, a $17 \times 17 \times 5$ domain decomposition is used. The results using a linear source approximation are presented in Figure 7-8 for the full core 3D BEAVRS benchmark, showing similar results to those with a flat source approximation.

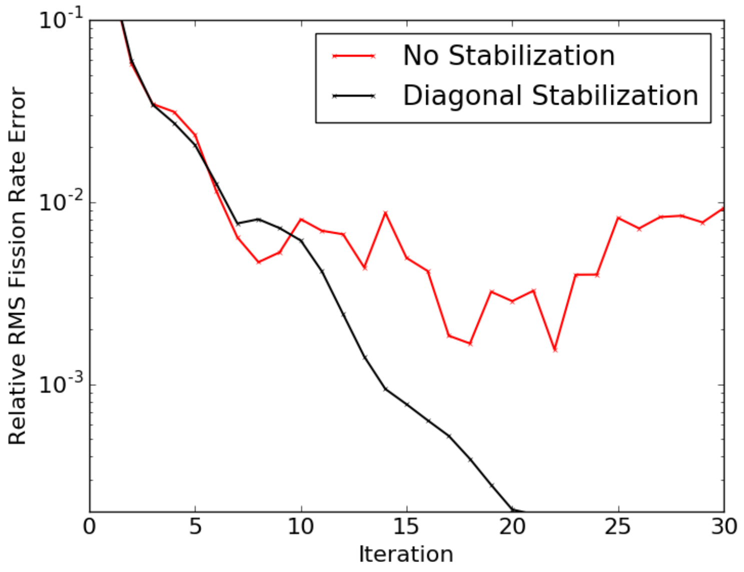


Figure 7-8: Convergence behavior of OpenMOC's linear source solver on the full core BEAVRS benchmark with and without Diagonal Stabilization ($\rho = 1/4$).

7.8 Conclusion

In this chapter, a theoretical analysis of source iteration convergence was presented which showed the possibility for source iteration instabilities with transport-corrected cross-sections. The theory was broadened to include source iterations without inner within-group scattering iterations. A new diagonal stabilization technique was presented which is substantially less conservative than previously proposed techniques. Realistic reactor physics problems were presented that showed instability with the source iteration process. The diagonal stabilization technique was shown to successfully stabilize the source iteration process without having much impact on convergence rate.

Results were also presented for convergence with CMFD acceleration which showed that CMFD acceleration without group collapse was able to stabilize the source iteration process. For CMFD acceleration with a collapsed group structure, the previously discussed stabilization techniques were necessary to ensure convergence. In the case of full core PWR problems, a collapsed group structure is often attractive to reduce run time, necessitating a stabilization method in order to converge.

Highlights

- With transport correction, source iteration is not guaranteed to converge
- Convergence issues were observed for models with deep reflectors, but never observed when using full-group CMFD acceleration
- A stabilization scheme, termed diagonal stabilization, is introduced which stabilizes source iteration convergence in all tested cases

Chapter 8

MOC Parameter Sensitivity Studies

In this chapter, sensitivities to both spatial mesh refinement and MOC ray refinement are studied in detail. Radial sensitivities, which have been studied extensively in 2D MOC are first discussed. Then, the axial sensitivities are studied. Both radial and axial sensitivities include mesh refinement as well as ray refinement studies with the OpenMOC linear source solver on BEAVRS geometries. These studies all involve models without control rod insertions. The axial sensitivity studies are then repeated for the case of a single assembly with a partially inserted control rod. In addition, the sensitivity of convergence to CMFD parameters and domain decomposition is also studied. This chapter concludes by presenting the chosen parameters to accurately and efficiently conduct PWR simulations with 3D MOC. The uninterested reader can skip to the conclusion of this chapter where the chosen parameters are presented.

8.1 Radial Sensitivity

Due to extensive experience with 2D MOC, the radial parameters required to achieve sufficient accuracy – both in mesh and ray refinement – are relatively well known. These requirements for linear source were thoroughly studied by Ferrer [41]. Based on these studies and CASMO-5 default parameters [40], the expected radial parameters needed to achieve a sufficiently accurate solution are given in Table 8.1. Each of the parameters listed in the table will be thoroughly discussed and undergo a refinement study.

Table 8.1: Expected radial MOC parameters to sufficiently resolve the MOC fission distribution using a linear source approximation

Number of Rings in Fuel and Moderator	1
Number of Sectors in Fuel	4
Number of Sectors in Moderator	8
Number of Sectors in Guide Tubes	8
Radial Ray Spacing	0.05 cm
Number of Azimuthal Angles	64
Reflector Mesh	3×3 cells per pin-cell mesh

8.1.1 Core Radial Mesh Refinement

First, the radial mesh within the core is studied. The core is composed of a lattice of assemblies. Therefore, the mesh refinement can be studied on a single assembly model. Since the axial dimensions should not largely impact the radial sensitivity, the short single assembly model is used, which is a 10 cm tall single assembly region without any grid spacers and reflective boundary conditions placed in both axial directions. This model is described in detail in Appendix E.2.6. This problem is simulated with the MOC ray parameters given in Table 8.2. These parameters are quite fine in the radial direction in order to accurately test the impact of the radial mesh. In the axial direction, the source height and axial ray spacing are quite coarse as the problem is uniform axially with no physical axial flux shape.

Table 8.2: MOC ray parameters for core mesh refinement studies

Radial Ray Spacing	0.025 cm
Number of Azimuthal Angles	128
Axial Ray Spacing	3.0 cm
Number of Polar Angles	10
Axial Source Height	10.0 cm

The radial profile of the fission rate distribution of the short assembly fuel rods is shown in Figure 8-1. The fission rates are calculated by running OpenMOC on the model with fine radial mesh. The radial distribution is formed by integrating the fission rates of all materials within each pin-cell.

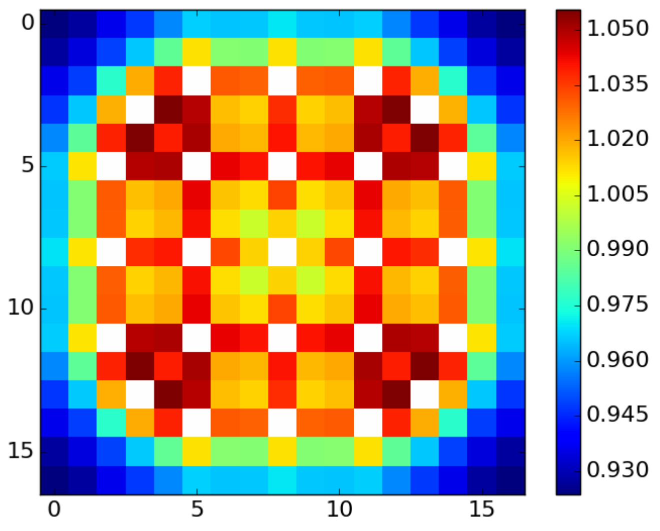


Figure 8-1: The fission rate distribution over pin-cells for the short single assembly model.

Typically, pin-cells are discretized into rings and sectors, as presented in Figure 8-2. Ring divisions help to capture radial variation within the pin-cell and sector divisions help to capture azimuthal variation.

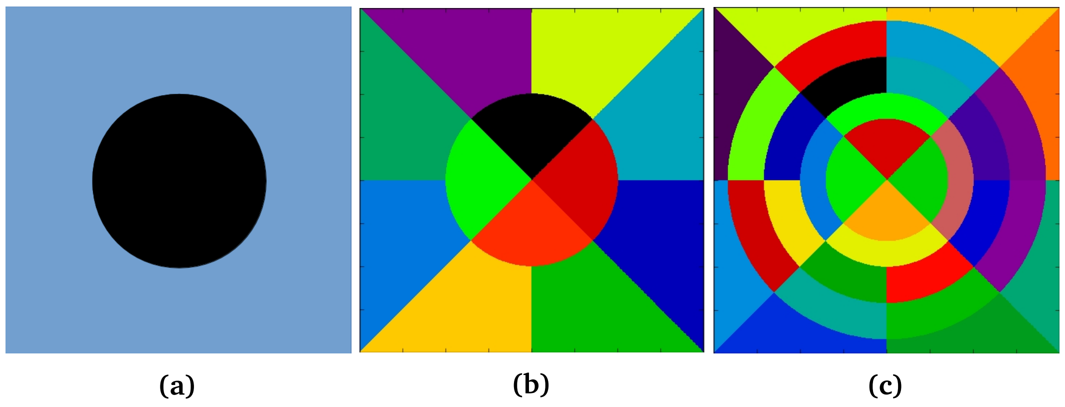


Figure 8-2: A simplified pin-cell (a) of just moderator and homogenized fuel is discretized (b) into 8 sectors in the moderator and 4 sectors in the fuel and further discretized (c) into 2 rings in the fuel and 3 rings in the moderator.

While the illustration in Figure 8-2 only shows fuel and moderator regions, realistic pin-cells contain fuel, gap, clad, and moderator. For mesh refinement, ring divisions are never added in gap and clad regions since they are so thin. In our studies, the sector divisions within fuel is also applied to the associated gap and clad regions.

Since both fuel pins and guide tubes exist in fuel assemblies, there are three main zones of interest that could be independently discretized: fuel pins, guide tubes, and moderator.

8.1.1.1 Ring Divisions

First, ring divisions are studied. The sensitivity studies of ring divisions are shown in Table 8.3. All fission rate errors are for a pin-wise mesh. Relative error in this thesis is always computed as

$$relative\ error = \frac{trial - reference}{reference}. \tag{8.1}$$

For all cases, the expected radial mesh parameters are used unless otherwise specified. For each zone, a separate parameter refinement is conducted using 1, 2, and 3 rings. The 3 ring case is always chosen as the reference. For the other radial mesh parameters, expected parameters are used (1 ring in all regions with 4 sectors in fuel and 8 sectors in the moderator and guide tube).

Table 8.3: MOC sensitivity to mesh refinement by radial rings in fuel, guide tube, and moderator regions

Region	No. of Rings	k_{eff}	k_{eff} Error	RMS Fission Rate Error	Max Fission Rate Error
Fuel	1	1.21568	0.9 pcm	0.002 %	0.003 %
Fuel	2	1.21569	0.4 pcm	0.001 %	0.002 %
Fuel	3	1.21569	–	–	–
Guide Tube	1	1.21569	0.1 pcm	0.002 %	0.004 %
Guide Tube	2	1.21569	<0.1 pcm	0.002 %	0.004 %
Guide Tube	3	1.21569	–	–	–
Moderator	1	1.21569	4.7 cm	0.003 %	0.007 %
Moderator	2	1.21564	-0.3 pcm	0.002 %	-0.004 %
Moderator	3	1.21564	–	–	–

These results show very little sensitivity. With a linear source approximation, gradients can be accurately captured without further radial discretization, causing ring divisions to be less important. Therefore, all tests in this study do not use ring discretizations.

8.1.1.2 Sector Divisions

Next, azimuthal sector discretization in analyzed. Since these discretizations largely account for azimuthal differences rather than radial gradients, they might still be needed with a linear source approximation. Again, the three regions (fuel, guide tubes, and moderator) are analyzed separately. Recall that gap and clad are discretized into the same number of sectors as the rod they surround.

For all of the cases, the RMS and maximum fission rate error is inconsequential, similar to that observed with discretization by rings. The maximum error occurred when comparing a case without any sector discretization for moderator, in which the RMS fission rate error was 0.1% and the maximum fission rate error was 0.3%, which are both fairly low. However, the effect on the eigenvalue k_{eff} is substantial. Therefore, the results for sector discretization focus on the impact on eigenvalue.

Similar to the approach taken for discretizing by rings, all mesh parameters are taken to be the expected parameters except the parameter being tested. All error estimates are formed relative to 16 sectors. Figure 8-3 shows the sensitivity to the sector discretization of fuel, guide tubes, and moderator.

These results show the moderator region is again the most sensitive. After 8 sectors in the moderator and 4 in fuel and guide tubes, very little sensitivity is observed. This would imply that requiring 8 sectors in the guide tubes may be excessive, but guide tubes are relatively sparse throughout the core so the increased discretization has little impact on computational requirements. Therefore, in order to be conservative, 8 sectors remains the choice for guide tube discretization.

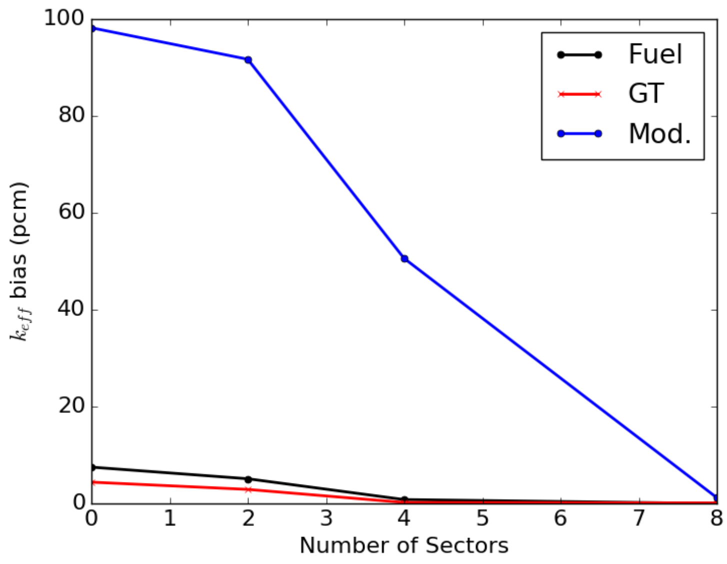


Figure 8-3: The bias in eigenvalue k_{eff} decreasing as the number of sector discretizations increased in fuel rod, guide tube, and moderator regions.

8.1.2 Reflector Radial Mesh Refinement

Ring and sector discretizations form a well-defined mesh refinement within the core. However, for full core problems, the radial water reflector also needs to be discretized. To simplify the sensitivity study, a uniform global mesh is overlaid across the reflector regions. In addition, the discretization is chosen to align with CMFD cell boundaries. In OpenMOC, CMFD cell boundaries naturally create mesh discretizations as cells are split at the mesh boundaries. Since a pin-cell width uniform CMFD mesh is chosen for acceleration of BEAVRS problems, the reflector is naturally discretized into pin-cell-sized regions. The radial mesh refinement then takes the form of a uniform mesh discretization within a pin-cell-sized region in which a square mesh refinement is chosen. Since each assembly contains a lattice of 17×17 cells and an assembly width is ≈ 21.5 cm (when including inter-assembly gap), the radial reflector mesh refinement takes the form of an $N \times N$ discretization of each 1.264 cm $\times$ 1.264 cm region within the reflector.

In order to conduct radial reflector mesh refinement sensitivity studies, a full core model is necessary. However, the full 3D BEAVRS model is computationally demanding. Therefore, the 2D BEAVRS model described in Appendix E.2.2 is chosen. This model represents a radial cut of the full core BEAVRS geometry that has been extruded 10 cm in height with reflective boundary conditions placed on the top and bottom of the geometry. With the height greatly reduced from the full 3D model, the computational requirements are far less. In addition, since this model contains no axial variation, the radial sensitivity can be tested more directly. The MOC parameters used in this sensitivity study are presented in Table 8.4.

Table 8.4: The MOC parameters used in the radial water reflector mesh refinement studies

Radial Ray Spacing	0.1 cm
Number of Azimuthal Angles	32
Axial Ray Spacing	1.5 cm
Number of Polar Angles	10
Axial Source Height	2.0 cm

The radial profile of the fission rate distribution of the 2D BEAVRS model is shown in Figure 8-4. The fission rates are calculated by running OpenMOC on the model with fine radial mesh in the reflector.

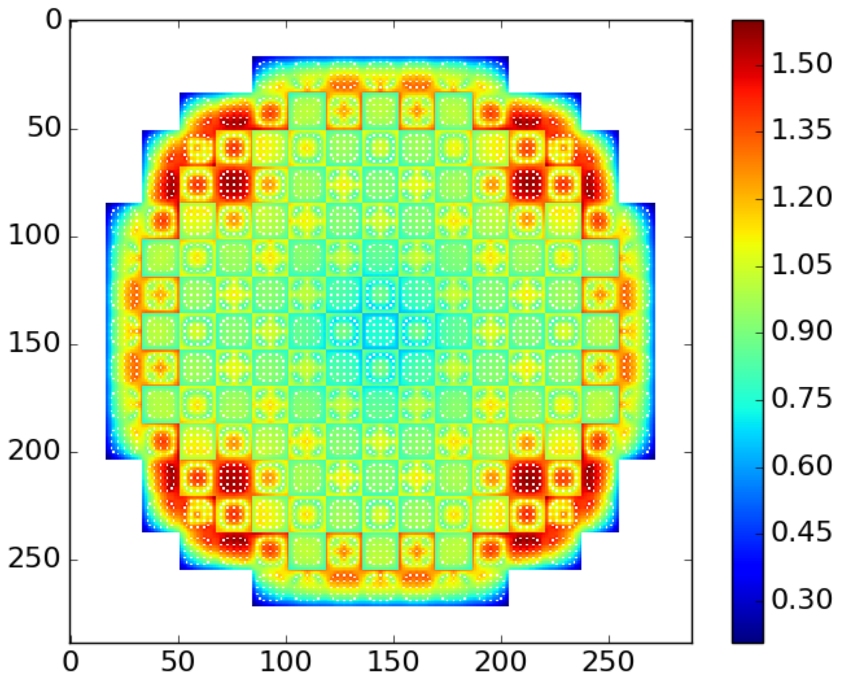


Figure 8-4: The fission rate distribution of pin-cells for the 2D BEAVRS model.

The results of the sensitivity study are presented in Figures 8-5 and 8-6 for eigenvalue and fission rate, respectively. All sensitivities are compared to a reference solution which uses a 5 × 5 radial reflector mesh discretization. The results show little sensitivity to the reflector mesh, likely since the linear source approximation is able to accurately capture gradients. At the expected 3 × 3 reflector mesh refinement, the eigenvalue bias is below 1 pcm and all relative fission rate errors are below 0.1%. Therefore, the 3 × 3 radial reflector mesh discretization is sufficient to accurately resolve the solution and is chosen for all further studies and results.

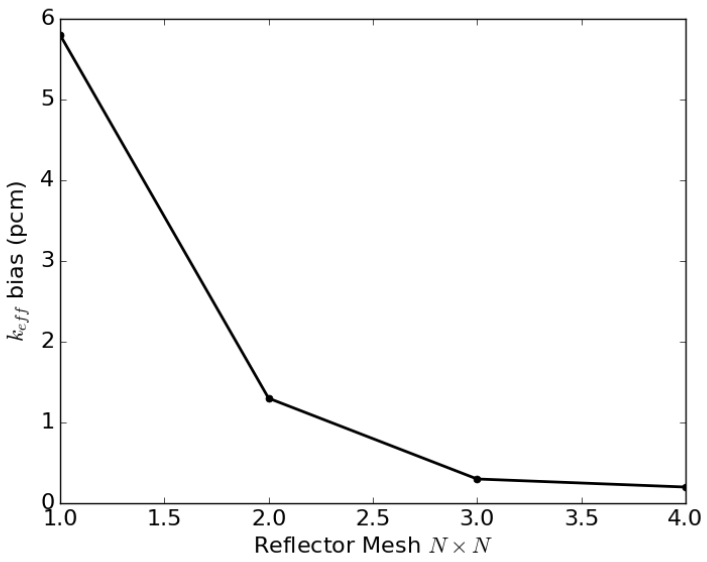


Figure 8-5: The bias in eigenvalue k_{eff} decreasing as reflector mesh of square discretization $N \times N$, where N is the number of mesh discretizations in each pin-cell width, is refined within the radial water reflector.

A depiction of the mesh refinement of a cutout encompassing both core and reflector regions is shown in Figure 8-7. The left side of the illustration shows the discretization of the core, in which pin-cells are carved into sectors. The baffle and water reflector on the right side are cut into rectangular regions formed by the superposition of material boundaries and the uniform mesh overlay.

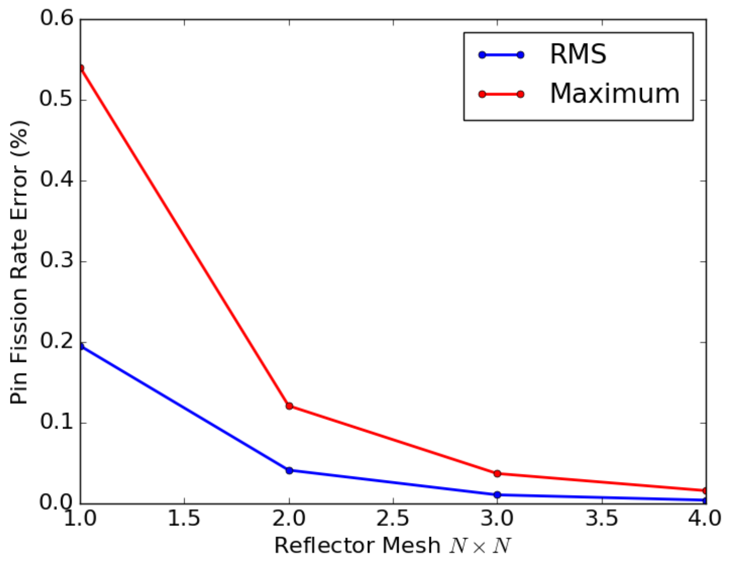


Figure 8-6: The relative pellet-wise fission rate error decreasing as reflector mesh of square discretization $N \times N$, where N is the number of mesh discretizations in each pin-cell width, is refined within the radial water reflector.

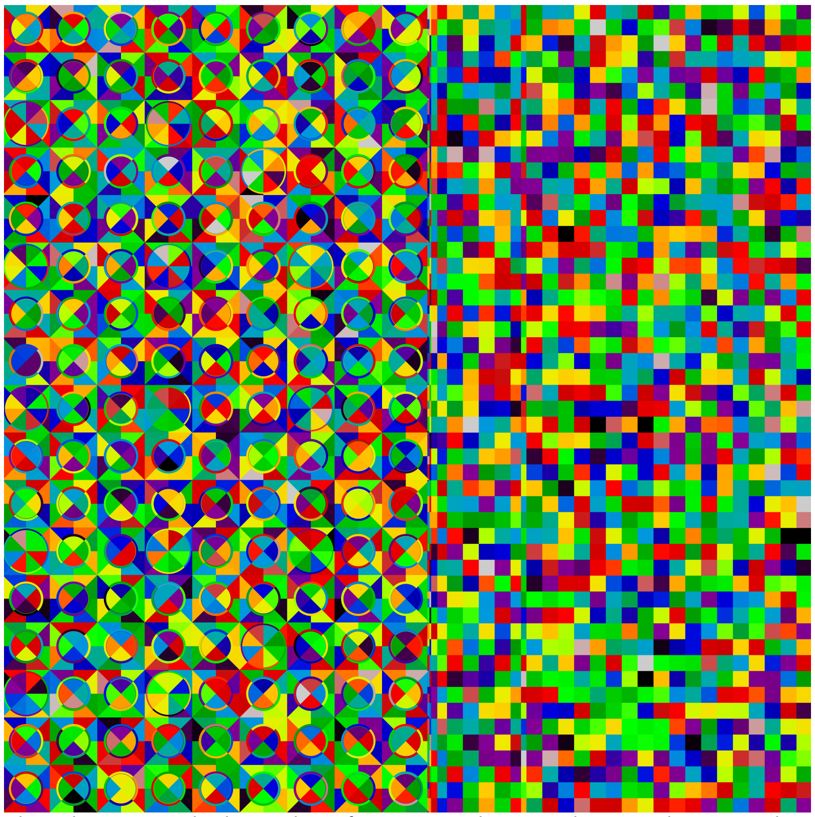


Figure 8-7: The chosen radial mesh refinement depicted near the periphery of the core, illustrating both core and reflector mesh.

8.1.3 Radial Ray Refinement

Next, the MOC radial ray parameters are investigated. The single assembly model is used in these tests, as described in Appendix E.2.3. This model has full axial height and includes grid spacers and axial water reflectors. For the radial discretization, the previous studies are used to inform the selected parameters, as given in Table 8.5. The number of azimuthal angles in $[0, 2\pi]$ and radial ray spacing required for accurate simulations are then tested using these assumed parameters. The radial mesh in the upper and lower water reflectors follows the same discretization present within the core. The radial profile of the fission rate distribution is very similar to the short assembly model, shown in Figure 8-1.

Table 8.5: MOC ray and mesh parameters for radial ray refinement studies

Number of Fuel Sectors	4
Number of Guide Tube Sectors	8
Number of Moderator Sectors	8
Axial Ray Spacing	0.75 cm
Number of Polar Angles	10
Axial Source Height	2.0 cm

8.1.3.1 Radial Ray Spacing Sensitivity

First, the radial ray spacing is tested. In these tests, the number of azimuthal angles is chosen to be 32. This is a factor of 2 away from the expected parameter, but still reasonably fine so it is assumed the radial ray spacing sensitivity is not significantly altered. The results of the sensitivity study are presented in Figures 8-8 and 8-9 for eigenvalue and fission rate, respectively. All errors and biases are reported relative to the case with 0.0125 cm radial ray spacing. These results show the expected radial ray spacing of 0.05 cm is sufficient to achieve less than 0.1% maximum pellet-wise fission rate error and less than 5 pcm bias.

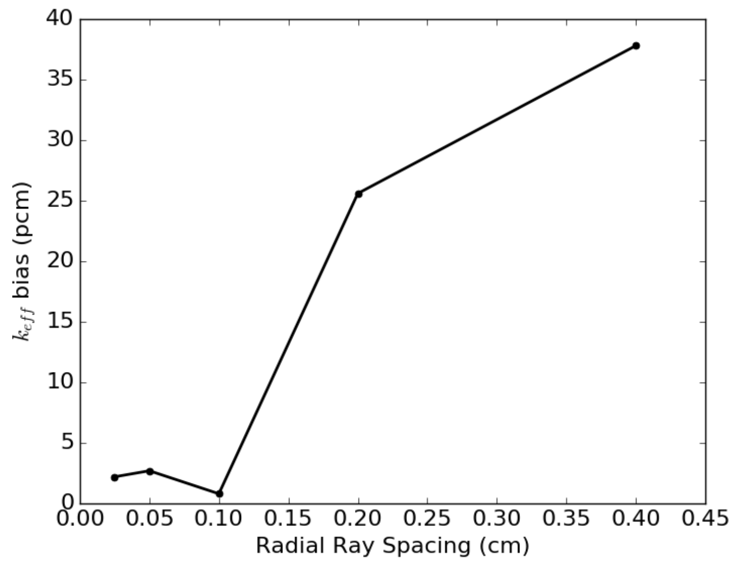


Figure 8-8: The bias in eigenvalue k_{eff} decreasing as radial ray spacing is refined.

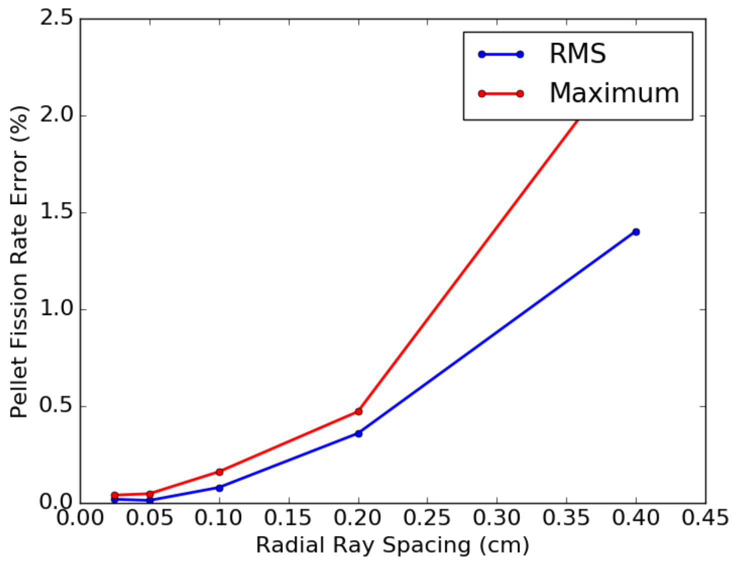


Figure 8-9: The relative pellet-wise fission rate error decreasing as radial ray spacing is refined.

8.1.3.2 Azimuthal Angle Sensitivity

The number of azimuthal angles is the last radial parameter to be tested. For these studies the radial ray spacing is chosen to be 0.1 cm, again a factor of 2 coarser than the expected parameter. The results of the sensitivity study are presented in Figures 8-10 and 8-11 for eigenvalue and fission rate, respectively. All errors and biases are reported relative to the case with 256 azimuthal angles.

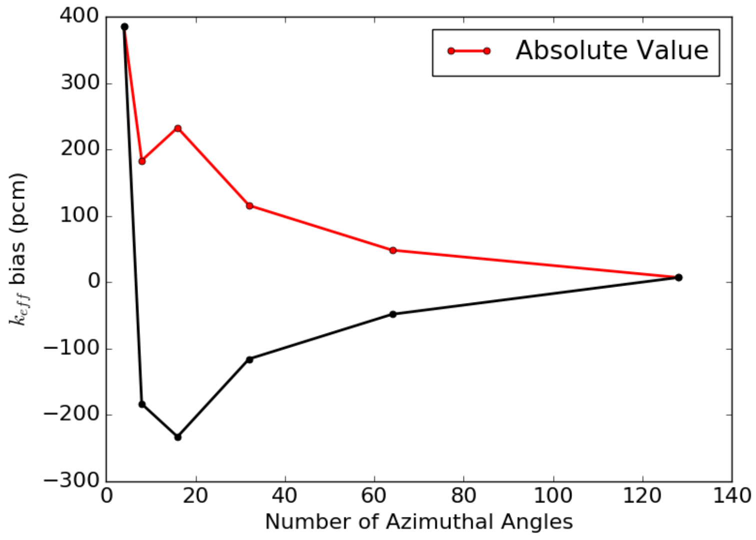


Figure 8-10: The bias in eigenvalue k_{eff} decreasing as the number of azimuthal angles is increased.

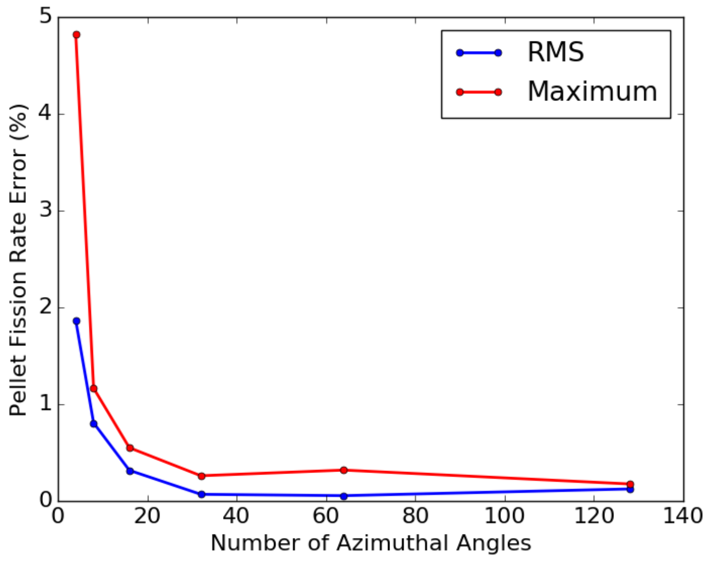


Figure 8-11: The relative pellet-wise fission rate error decreasing as the number of azimuthal angles is increased.

Here there seems to be significant sensitivity to the number of azimuthal angles. At the expected 64 azimuthal angles in $[0, 2\pi]$, there is a 48.3 pcm bias. However, the error in pellet-wise reaction rates is significantly less with the 64 azimuthal angle case producing less than 0.1% RMS error and less than 0.5% maximum error. Therefore, the expected 64 azimuthal angles are assumed to be sufficient for accurately resolving the fission rate distribution.

8.2 Axial Sensitivity

The previous section studied radial sensitivities, which are known quite well due to extensive experience with 2D MOC. Therefore, the studies were a verification that the expected parameters were sufficient to accurately simulate typical geometries and materials found in PWRs.

In this section, the axial sensitivities are studied, for which there is much less collective experience. 3D MOC solvers have only recently been investigated in great detail. Therefore, rather than verifying a set of MOC parameters to be sufficient for accurate simulations, a search is required to find the accurate parameters. This requires some metric for determining whether a simulation is sufficiently accurate.

The following criteria is selected to constitute an accurate simulation: less than 1.0% RMS pellet fission rate error, less than 3.0% maximum pellet fission rate error, and less than 20 pcm bias on eigenvalue compared with a converged reference solution. The pellet fission rate is defined to be the fission rate within a given 2.0 cm tall fuel pin region. The reference solution for every case is chosen by further refining the parameter of interest, similar to the radial studies. This allows each parameter to be independently studied.

In this study, the single assembly model detailed in Appendix E.2.3 is chosen which contains axial water reflectors and grid spacers. It does not contain any inserted rods. Later, a single assembly with inserted rods is analyzed to determine the MOC parameters when there are significant axial gradients. The axial profile of the fission distribution for the single assembly model without rod insertions is presented in Figure 8-12, formed from an OpenMOC run with fine MOC parameters and radially integrating fission rates. Notice the fission distribution dips around grid spacers.

This section focuses on understanding the sensitivity to three axial parameters: the axial source height, the axial ray spacing, and the number of polar angles in $[0, \pi]$. For

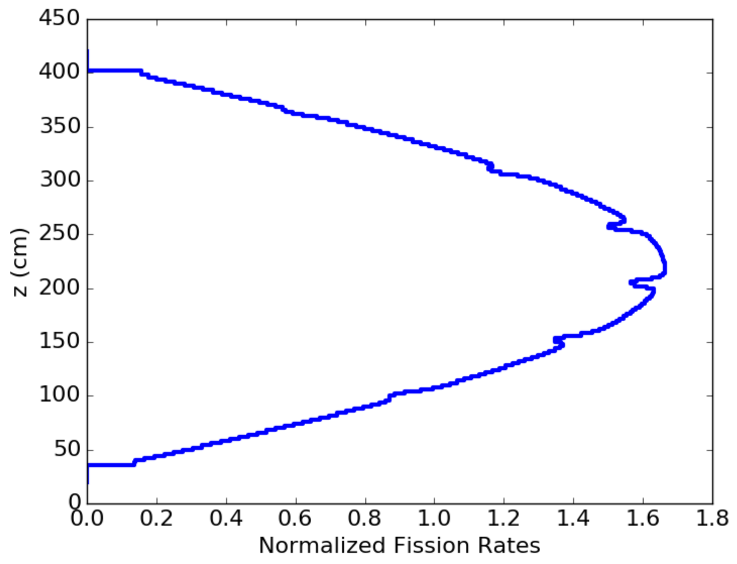


Figure 8-12: The axial fission rate distribution of the single assembly model formed from radially integrating reaction rates in each 2 cm tall region.

the axial source height sensitivity tests, a uniform axial mesh is imposed in order to simplify the study. Due to the structure of the test geometries presented in Appendix E, all material boundaries occur at even intervals. Therefore, to impose a uniform axial mesh, the maximum axial source height is 2.0 cm. The axial ray spacing and polar angle tests are conducted in a similar fashion to the radial ray spacing and azimuthal angle sensitivity tests. For all tests, the radial parameters presented in Table 8.6 are used. The radial mesh in the axial water reflectors follows the same discretization as that present within the core.

Table 8.6: MOC ray and mesh parameters for radial ray refinement studies

Number of Fuel Sectors	4
Number of Guide Tube Sectors	8
Number of Moderator Sectors	8
Number of Azimuthal Angles	32
Radial Ray Spacing	0.1 cm

8.2.1 Axial Source Height Sensitivity

First, the axial source height sensitivity study is conducted. For these tests, 10 polar angles and an axial ray spacing of 0.09375 cm (3/32 cm) are selected. The reference case has an axial source height of 0.25 cm. The results are presented in Figures 8-13 and 8-14 for eigenvalue and fission rate, respectively.

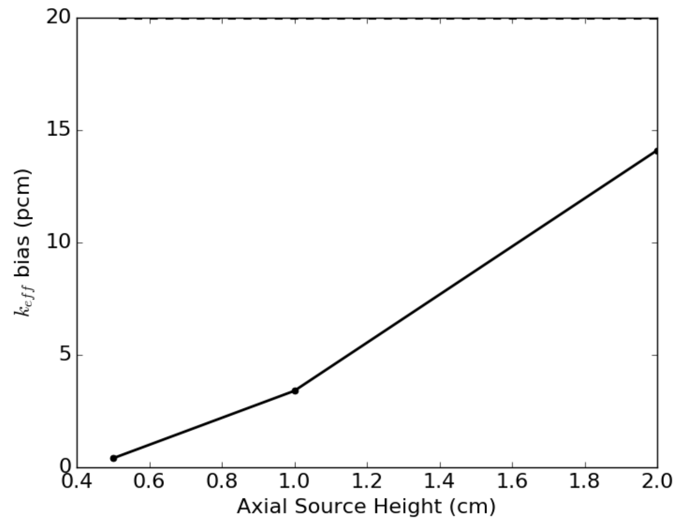


Figure 8-13: The bias in eigenvalue k_{eff} decreasing as axial source height is refined.

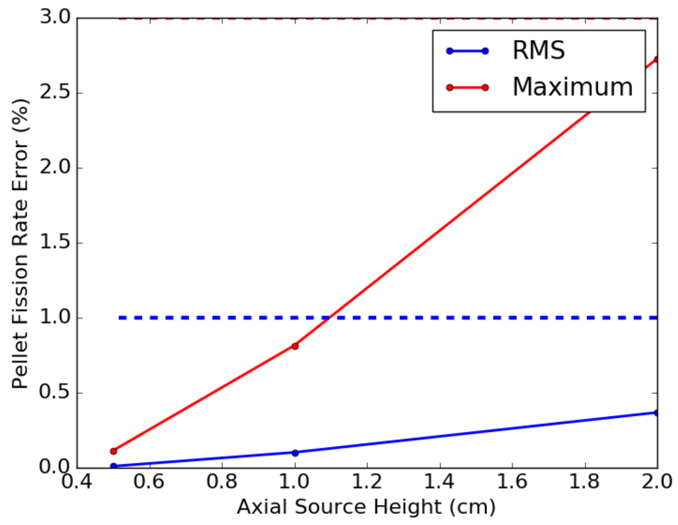


Figure 8-14: The relative pellet-wise fission rate error decreasing as axial source height is refined.

Notice that there is a sensitivity to axial source height but with 2.0 cm source height, the accuracy criteria is satisfied. However, if less than 1.0% *maximum* fission rate error were required, the axial source height would need to be 1.0 cm.

8.2.2 Axial Ray Spacing Sensitivity

Next, the axial ray spacing sensitivity is studied. For these tests, 10 polar angles and an axial source height of 2.0 cm are selected. Since, the axial ray spacing should be less than the axial source height to ensure at least one crossing of each source region per angle, the coarsest axial ray spacing is chosen to be 1.5 cm. Then the ray spacing is halved each time the ray spacing is refined. The reference is chosen to be 0.09375 cm (3/32 cm). The results are presented in Figures 8-13 and 8-14 for eigenvalue and fission rate, respectively. These results show that 1.5 cm axial ray spacing is sufficient to satisfy the accuracy criteria. However, if less than 1.0% maximum fission rate error is required, the axial ray spacing would need to be 0.75 cm.

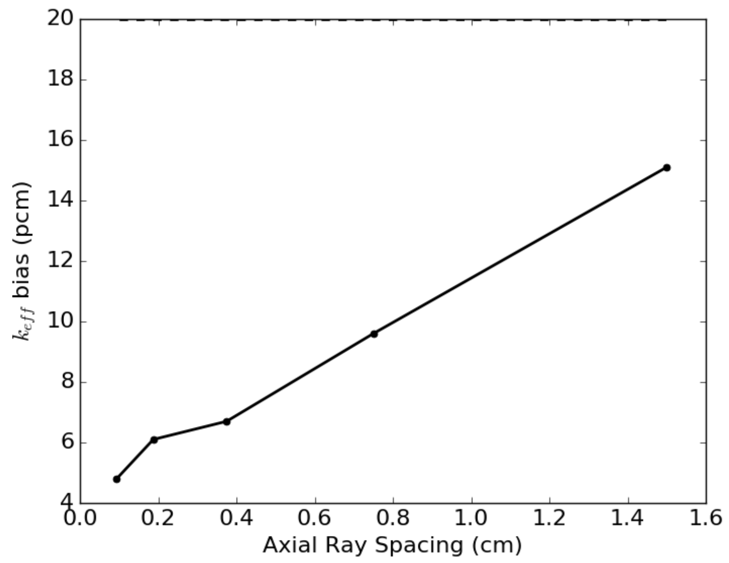


Figure 8-15: The bias in eigenvalue k_{eff} decreasing as axial ray spacing is refined.

This result is quite similar to the axial source height sensitivity whereby a refinement of the parameter may be useful. Since it is not possible to refine axial source height further with an axial ray spacing of 1.5 cm while ensuring each axial source region is traversed, the axial ray spacing is chosen to be 0.75 cm to allow for a source height

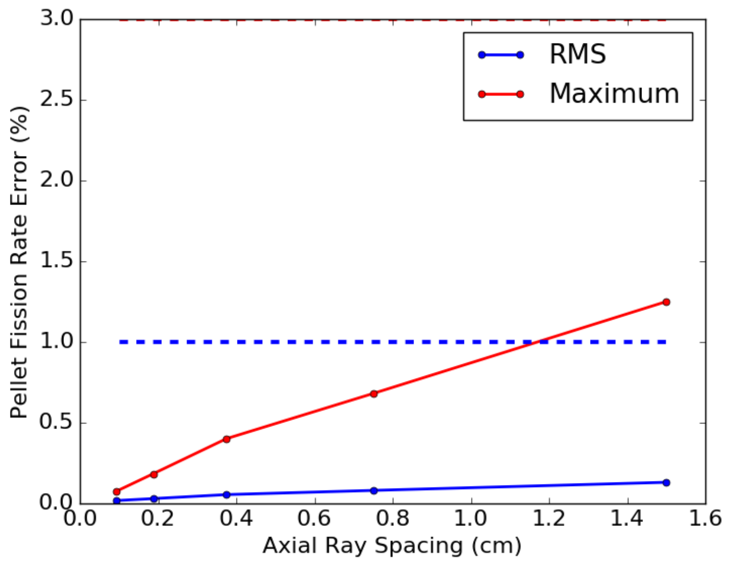


Figure 8-16: The relative pellet-wise fission rate error decreasing as axial ray spacing is refined.

refinement if necessary.

8.2.3 Polar Angle Sensitivity

Lastly, the polar angle sensitivity study is conducted using an axial ray spacing of 0.25 cm and an axial source height of 2.0 cm. This fine axial ray spacing allows the trend to be less sensitive to small perturbations in track laydown when changing polar angles, causing the trend to be more directly visible. All results use the Gauss-Legendre polar quadrature. The reference is chosen to be 32 polar angles in $[0, \pi]$. The results are presented in Figures 8-17 and 8-18 for eigenvalue and fission rate, respectively. These results show that only 6 polar angles are necessary to achieve the accuracy criteria. However, 2D MOC simulations with Gauss-Legendre quadrature typically use a minimum of 5 polar angles in $[0, \pi/2]$, equivalent to 10 polar angles for 3D MOC in $[0, \pi]$. Therefore, 10 polar angles are assumed to be necessary.

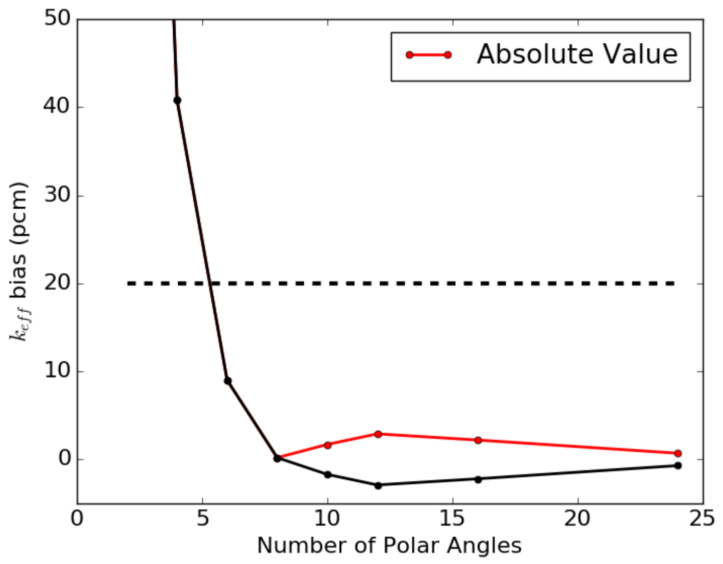


Figure 8-17: The bias in eigenvalue k_{eff} decreasing as the number of polar angles is increased.

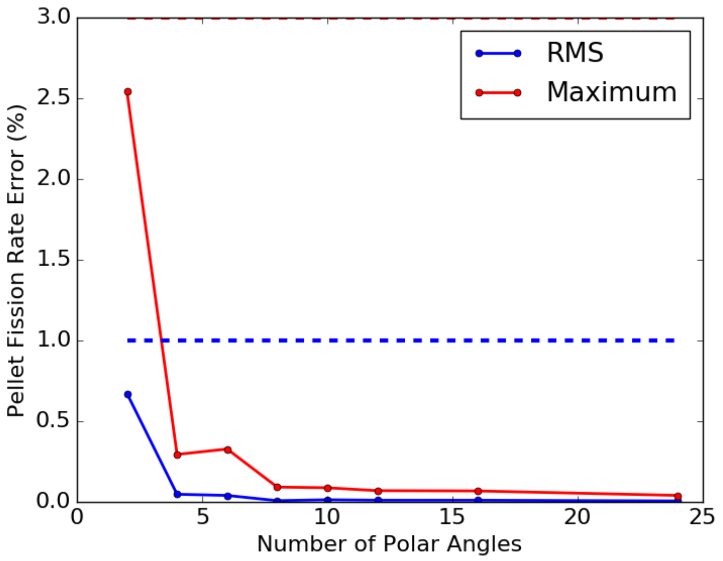


Figure 8-18: The relative pellet-wise fission rate error decreasing as the number of polar angles is increased.

8.3 Axial Sensitivity on a Rodded Assembly

In the previous section, the axial MOC parameters were studied for a single assembly problem with no control rod insertions, allowing for smooth axial gradients. When control rods are inserted, the gradients can become significant. To observe this effect, the rodded single assembly model discussed in Appendix E.2.7 is selected. In this problem, a rod is halfway inserted into the assembly, creating a significant distortion in the power profile. The axial profile of this model is presented in Figure 8-19, again formed from an OpenMOC run with fine MOC parameters and radially integrating fission rates.

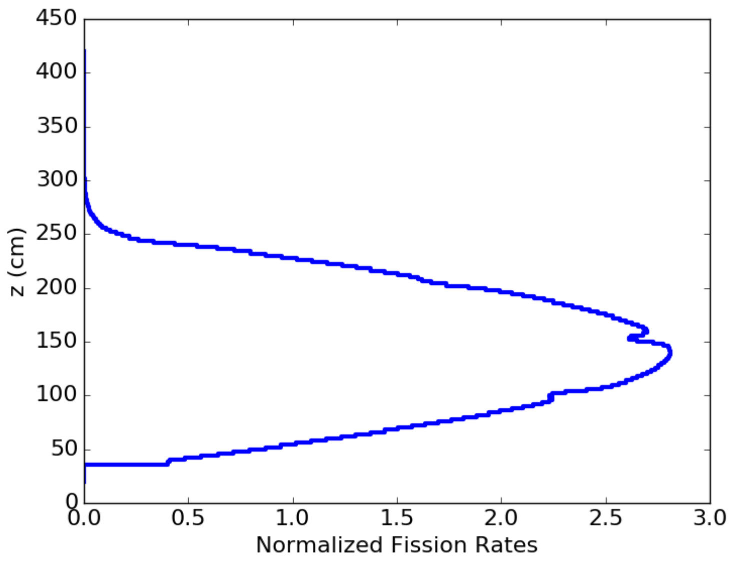


Figure 8-19: The axial fission rate distribution of the rodded single assembly model formed from radially integrating reaction rates in each 2 cm tall region.

The presence of significant gradients mean finer axial parameters may be required. Therefore, the axial sensitivity study is repeated for the rodded single assembly test problem. Again, the radial parameters described in Table 8.6 are selected. Since the fission distribution drops significantly past the control rod tip, fission rate error comparisons in this section only consider the region below the axial height of 250 cm.

8.3.1 Axial Source Height Sensitivity

First, the axial source height is again analyzed using 10 polar angles and an axial ray spacing of 0.09375 cm. The reference again has 0.25 cm axial source height. The results are presented in Figures 8-17 and 8-18 for eigenvalue and fission rate, respectively. Now, a finer axial source discretization is required, with a source height of 1.0 cm necessary to achieve the accuracy criteria.

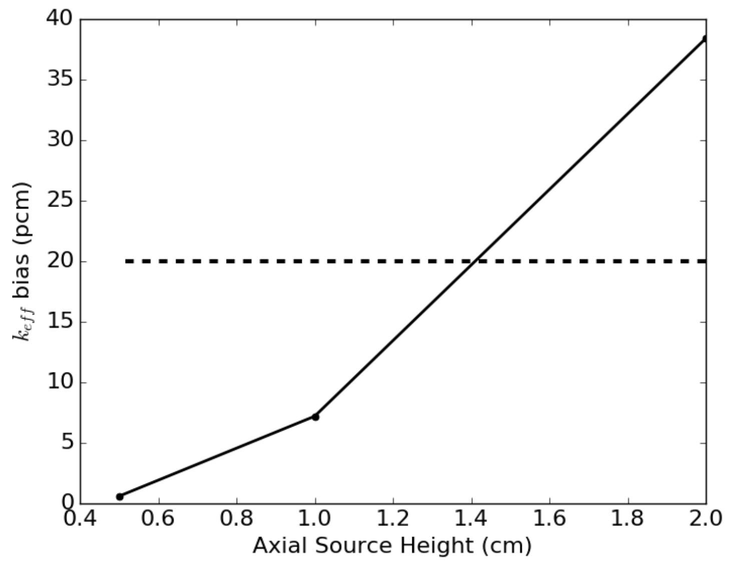


Figure 8-20: The bias in eigenvalue k_{eff} decreasing as axial source height is refined.

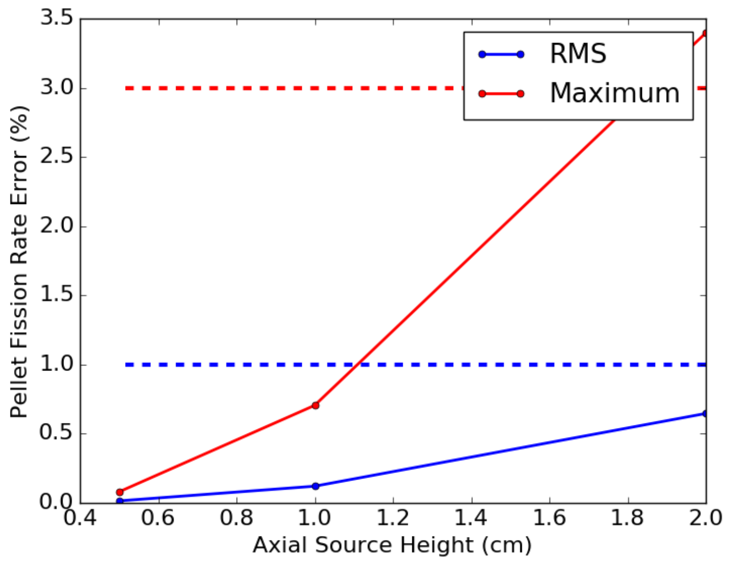


Figure 8-21: The relative pellet-wise fission rate error decreasing as axial source height is refined.

8.3.2 Axial Ray Spacing Sensitivity

Next, the axial ray spacing is analyzed using 10 polar angles and a source height of 0.5 cm. The reference axial ray spacing is chosen to be 0.023 cm (3/128 cm). Since the source height has been decreased from 2.0 cm to 0.5 cm in comparison with the tests in Section 8.2.2, the coarsest axial ray spacing present in the previous tests that still traverses every axial source region is 0.375 cm (3/8 cm). Again, the axial ray spacing is halved at each refinement. The results are presented in Figures 8-22 and 8-23 for eigenvalue and fission rate, respectively. An axial ray spacing of 0.375 cm (3/8 cm) is sufficient to achieve less than 1% maximum fission rate error.

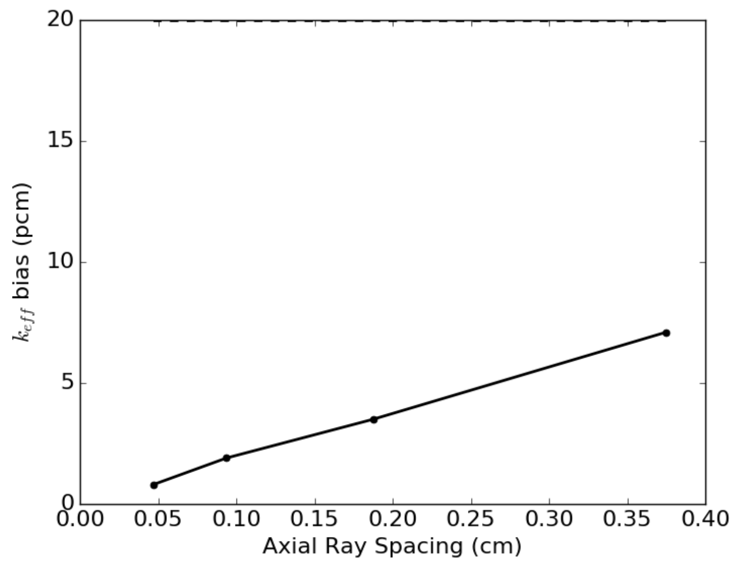


Figure 8-22: The bias in eigenvalue k_{eff} decreasing as axial ray spacing is refined.

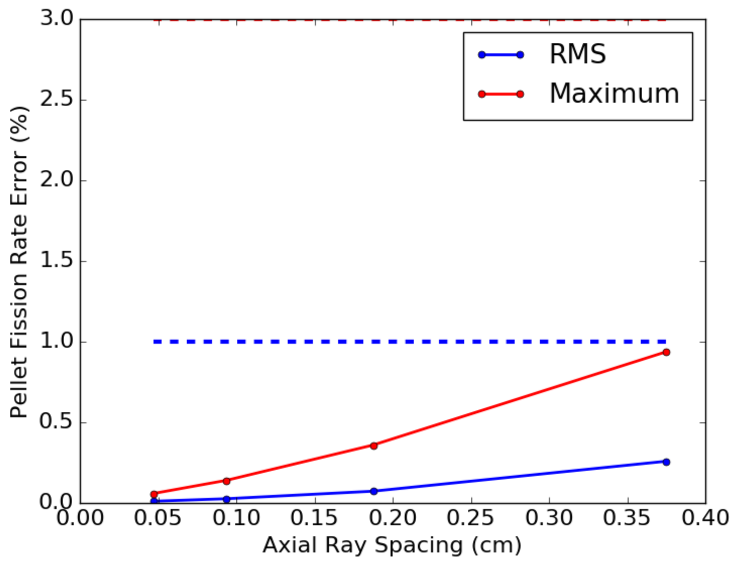


Figure 8-23: The relative pellet-wise fission rate error decreasing as axial ray spacing is refined.

8.3.3 Polar Angle Sensitivity

Lastly, the polar angle sensitivity is conducted with the same parameters as the case without rod insertions in Section 8.2.3: 0.25 cm axial ray spacing and 2.0 cm axial source height. Again, the reference is 32 polar angles. While the axial source height is coarser than necessary to accurately converge the problem, it is assumed that the sensitivity is similar with the necessary parameter. The results are presented in Figures 8-22 and 8-23 for eigenvalue and fission rate, respectively, showing very little difference from the case without rod insertions. Again, 6 polar angles are sufficient to meet the criteria but 10 polar angles are selected due to the 2D MOC considerations outlined in Section 8.2.3.

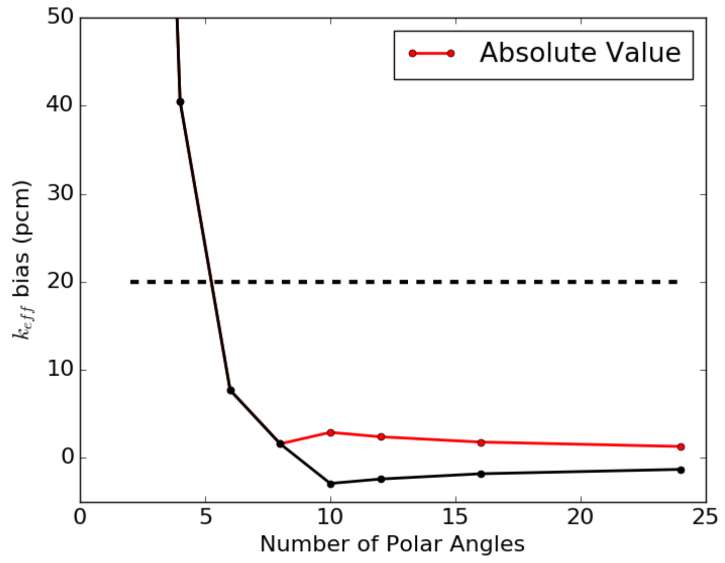


Figure 8-24: The bias in eigenvalue k_{eff} decreasing as the number of polar angles is increased.

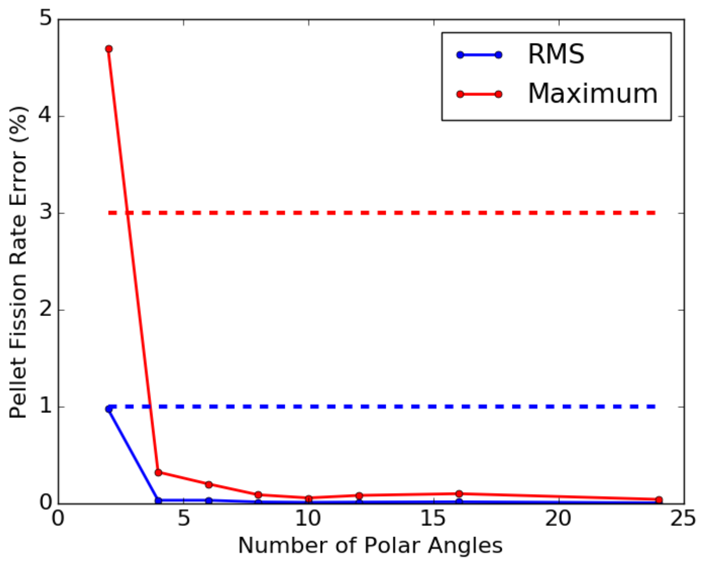


Figure 8-25: The relative pellet-wise fission rate error decreasing as the number of polar angles is increased.

8.4 Comparison with Flat Source MOC

In the previous sections, the radial and axial parameters required to achieve accurate 3D MOC solutions were investigated. In this section, the parameters which were found to be sufficient in accurately simulation PWR problems – and presented later during the conclusion in Table 8.7 – are used to simulate the single assembly model described in Appendix E.2.3 for both linear source and flat source solvers. Since the parameters were derived with the linear source parameters, there should be some bias introduced with the flat source solver.

The results show a bias of 41 pcm was incurred with the flat source solver compared with the linear source solution. Additionally, the pellet-wise RMS relative fission rate difference was 2.1% and the maximum relative fission rate difference was 12.9%.

A reference OpenMC solution was generated for the problem to compare the accuracy of flat and linear solutions. The reference OpenMC results simulated 400 batches of neutrons (300 inactive, 100 active) with 10^7 particles per batch on the single assembly model. The axial fission rate error relative to the OpenMC reference solution is presented in Figure 8-26 where the fission rates in each axial interval are formed by radially integrating all region-wise fission rates within the axial interval.

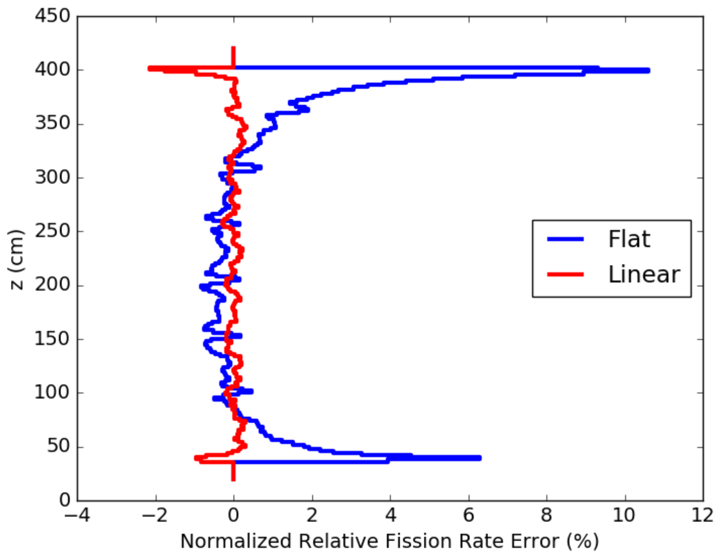


Figure 8-26: The axial fission rate error of flat source and linear source simulations relative to an OpenMC reference solution on the single assembly test problem.

From the axial fission rate error distribution in Figure 8-26, the largest errors occur near the axial water reflectors. This is due to the flat source approximation not being able to capture the strong axial gradients that occur near the reflectors. In order to achieve comparable accuracy using the flat source solver, a significantly finer source discretization in the axial direction would likely need to be used, causing the axial ray spacing to also be further refined.

8.5 CMFD Acceleration

In all of the previous sections of this chapter, only solution accuracy was investigated as a function of MOC parameters. CMFD acceleration is often necessary to achieve reasonable convergence rates, as detailed in Appendix B. Therefore, the sensitivity of convergence rate to CMFD parameters is discussed in this chapter. It is important to note that the acceleration does not impact solution accuracy.

CMFD is implemented in OpenMOC with uniform mesh on any reduced group structure. Therefore, the important parameters for defining CMFD acceleration are the mesh dimensions and the number of energy groups. For the PWR problems investigated in this thesis, pin-cell mesh is always chosen in the radial plane. This is chosen based on extensive experience in 2D MOC. Therefore, this section focuses on understanding the sensitivity to two remaining parameters: the axial mesh and the number of energy groups used in the CMFD solver. In order to decrease run-time requirements of the CMFD solver, the parameters should be chosen to be as coarse as possible while maintaining an optimal convergence rate.

For this investigation, the single assembly model described in Appendix E.2.3 is used with the MOC parameters determined in the previous sections to obtain reasonable solution accuracy, presented later in Table 8.7. For all the trials, MOC diagonal damping is applied, as discussed in Chapter 7, with $\rho = 1/4$. A CMFD damping factor of 0.7 is also applied for all trials.

8.5.1 Axial Mesh Sensitivity

First, the axial mesh sensitivity is investigated. Since a uniform mesh is required by the CMFD solver, the number of axial CMFD cells N_Z uniquely defines the axial mesh. The axial height of the single assembly problem is 400 cm and the axial source height is 2.0 cm. When CMFD boundaries intersect source regions, the source regions are split such that each source region belongs to uniquely one CMFD cell. Therefore, in order to have the same source region discretization for all tested cases, the number of axial cells N_Z is chosen such that $200/N_Z$ is an integer. The convergence results are presented in Figure 8-27 for a variety of CMFD axial mesh cells.

The results show that further mesh discretization improves convergence rate all the way to the maximum number of axial CMFD cells ($N_Z = 200$) in which each CMFD spans only the height of one MOC source region. Therefore, 200 CMFD cells should be used in all calculations. At just 25 axial CMFD cells, it is not clear that the solution will even converge, as the residual does not drop significantly in the first 50 iterations.

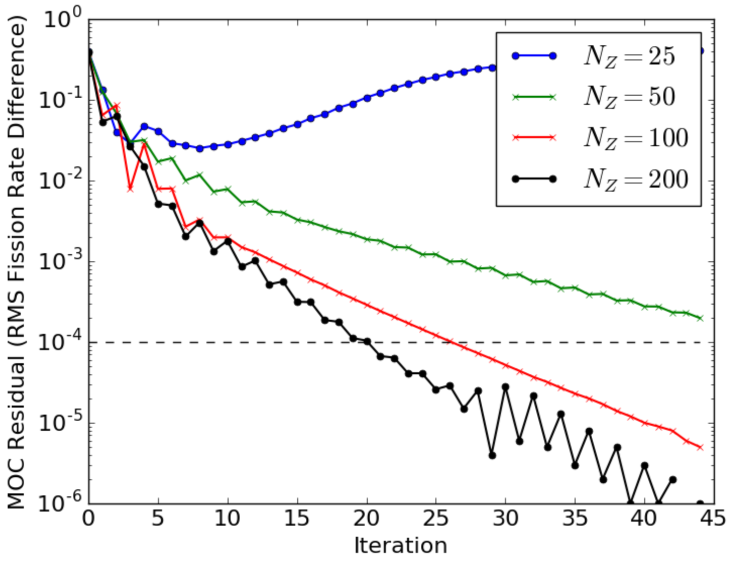


Figure 8-27: Convergence with a variety of CMFD axial mesh cells, N_Z.

8.5.2 Energy Group Sensitivity

Next, the sensitivity to the number of CMFD energy groups is studied. Since the number of MOC energy groups is 70, the maximum number of CMFD groups is 70. For less than 70 energy groups, certain groups are combined. There are many configurations in which the group structure can be collapsed into a reduced CMFD group structure. In OpenMOC, any collapse is possible, but the CASMO-4 group structures [63] are implemented for convenience. These group structures can be found in Appendix C. Using these reduced group structures and varying the number of CMFD groups G_C, the sensitivity of convergence rate to the CMFD group structure is tested. The results are shown in Figure 8-28.

Notice that the convergence history is not significantly different for 8 or more CMFD groups. However, when the number of CMFD groups is reduced beyond 8, the convergence suffers. With one CMFD group, the residual does not drop significantly during the first 50 iterations. Therefore, this study shows that 8 CMFD groups should be used in simulation as it captures the optimal convergence rate with the minimum number of energy groups.

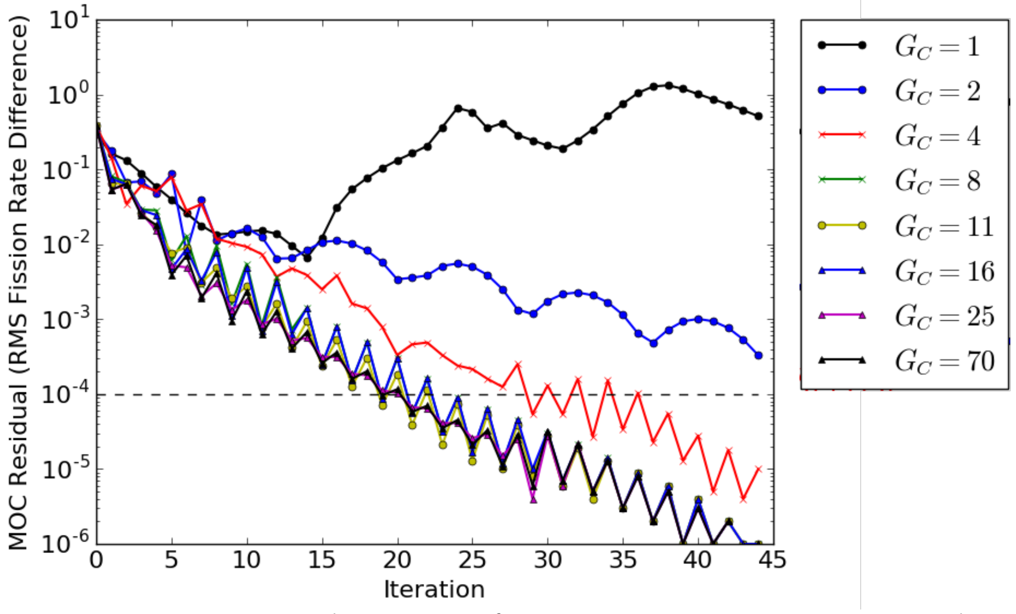


Figure 8-28: Convergence with a variety of CMFD energy group structures with number of groups G_C.

8.6 Domain Decomposition

In Chapter 6, it was noted that domain decomposition could slow the convergence rate since angular fluxes are lagged at domain interfaces. Here, the sensitivity to axial domain decomposition is studied on a single assembly model. The MOC and CMFD parameters presented later during the conclusion in Table 8.7 are used in this study, except the number of azimuthal angles is reduced to 32 and the radial ray spacing is coarsened to 0.1 cm in order to run in reasonable time without domain decomposition. The sensitivity of convergence to the number of axial domains D_Z is shown in Figure 8-29.

The results show a slight but constant degradation in convergence rate as the assembly is decomposed into more axial domains up to 100 axial domains. At 200 axial domains, where each domain only covers one axial source region, the convergence rate is significantly degraded. However, a very fine axial decomposition is not intended. In order to most effectively reduce the memory storage per domain, nearly cubic domains should be created, as on-node computational work scales with domain volume but on-node memory storage scales with domain surface area. For the full core BEAVRS benchmark, a radial domain decomposition of single assemblies is intended, which have

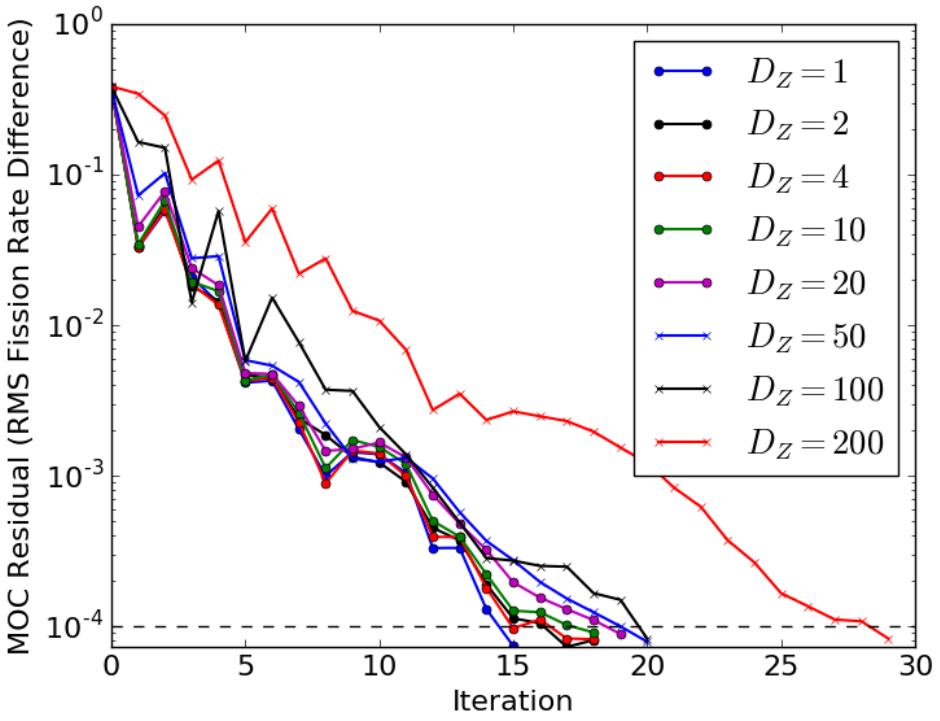


Figure 8-29: Convergence with a $1 \times 1 \times D_Z$ domain decomposition used on the single assembly model.

width approximately 21.42 cm. Since the height of the simulated BEAVRS model is 400 cm, an axial domain decomposition of 20 axial domains would lead to nearly cubic domains, and is the intended domain decomposition for the BEAVRS benchmark in this thesis. At 20 axial domains, the single assembly convergence is only slightly degraded from the single domain case.

8.7 Conclusion

In this chapter, the sensitivity of solution accuracy to MOC parameters was investigated in great detail. In addition, the convergence of CMFD was studied as a function of CMFD energy group structure and axial mesh. From these studies, the parameters given in Table 8.7 are chosen to accurately and efficiently simulate PWR problems using OpenMOC's 3D MOC linear source solver. Using the flat source solver with these parameters incurs significant error.

Table 8.7: MOC ray and mesh parameters determined to accurately and efficiently simulate PWR problems

Radial Ray Spacing	0.05 cm
Axial Ray Spacing	0.75 cm
Number of Azimuthal Angles	64
Number of Polar Angles	10
Number of Fuel Sectors	4
Number of Guide Tube Sectors	8
Number of Moderator Sectors	8
Axial Source Height	2.0 cm
Radial Reflector Mesh	3×3 cells per pin-cell mesh
CMFD Cell Height	2.0 cm
Number of CMFD Energy Groups	8

Highlights

- The presence of rod insertions requires axial source height and axial ray spacing to be refined, but there is little polar angle sensitivity.
- Significant error is introduced when the MOC parameters necessary for accuracy with the linear source solver are used with the flat source MOC solver
- For optimal CMFD convergence, the CMFD axial mesh should be the same as the MOC axial mesh, but the number of CMFD energy groups can be reduced to 8 from the 70 group MOC structure without reduction in convergence rate

Chapter 9

Full Core Results

The objective of this thesis is to efficiently simulate a full core LWR using 3D MOC, converged in space and angle. In this chapter, the OpenMOC implementation described in previous chapters is used to directly simulate the full core BEAVRS benchmark. In past presentations, the 3D MOC solver implemented in this thesis has been verified on a variety of simpler, conventional reactor physics models [56,64]. The BEAVRS benchmark represents a far more computationally demanding challenge, especially with a 70 group cross-section library. The computed OpenMOC solution using 3D MOC is compared with the reference OpenMC Monte Carlo solution. Then, the computational performance and profile of OpenMOC on the full core problem is analyzed. To ensure the chosen MOC parameters are sufficient for spatial and angular convergence, a parameter refinement study is also conducted. The chapter concludes by summarizing and highlighting important results.

9.1 Comparison with OpenMC

In Chapter 8, parameter refinement studies were conducted for each MOC parameter. Using the MOC parameters sufficient to accurately and efficiently resolve pellet-wise fission rates, as presented in Table 8.7 with the mesh shown in Figure 8-7, the BEAVRS benchmark is simulated on the Mira partition of the Argonne BlueGene/Q supercomputer using the linear source 3D MOC solver in OpenMOC. The resulting radially and axially integrated reaction rates are presented in Figure 9-1 and Figure 9-2.

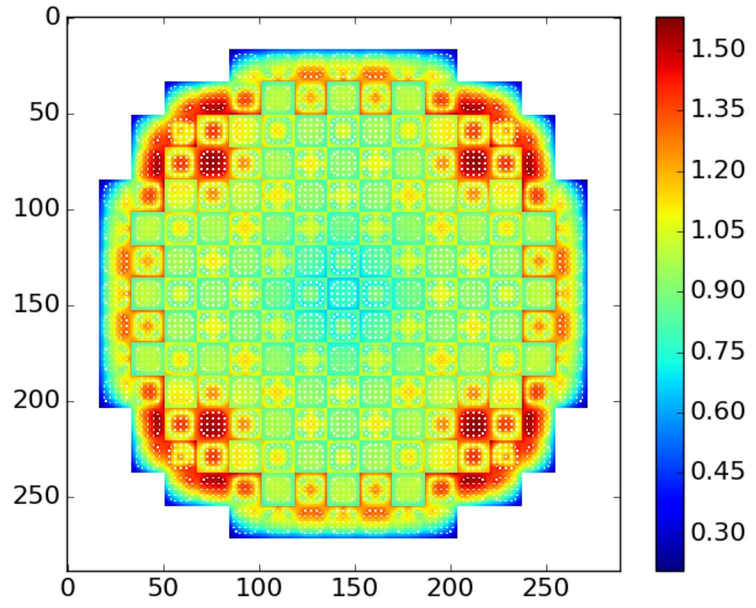


Figure 9-1: Pin-wise radial fission rate distribution for the BEAVRS benchmark formed by OpenMOC with reaction rates axially integrated.

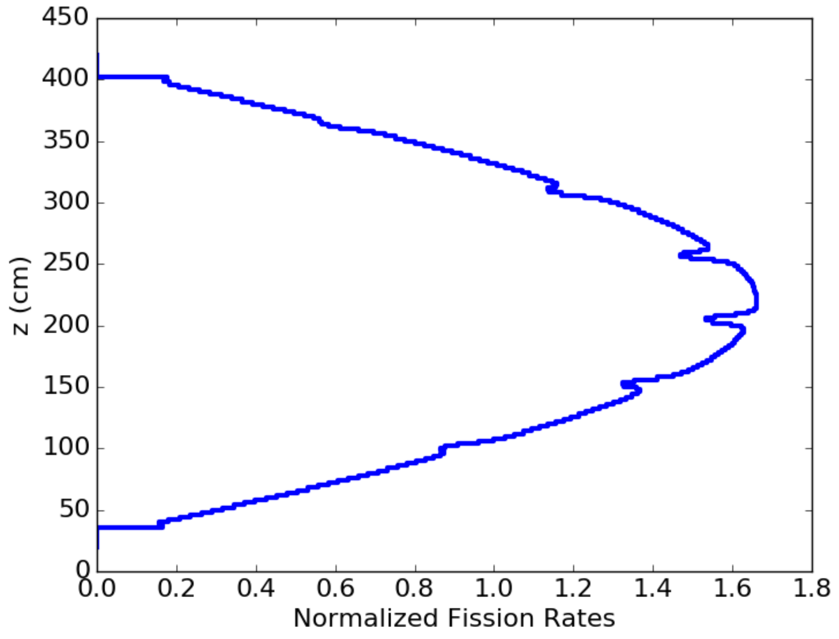


Figure 9-2: Axial fission rate distribution for the BEAVRS benchmark formed by OpenMOC with reaction rates radially integrated.

The OpenMOC results are compared with the OpenMC Monte Carlo solution from which the multi-group cross-sections were derived. The Monte Carlo simulation used 400 batches (300 inactive, 100 active) with 2×10^8 particles per batch. A comparison of the OpenMOC and OpenMC solutions is presented in Table 9.1 in which fission rates are calculated on a pellet-wise scale (2.0 cm axial height).

Table 9.1: Simulation accuracy of OpenMOC relative to an OpenMC reference solution

	k_{eff} eigenvalue	Avg. Fission Rate Std. Dev.	RMS Fission Rate Error	Max Fission Rate Error
OpenMC	0.99927 +/- 1×10^{-5}	1.82%	–	–
OpenMOC	0.99677	–	2.14%	7.52%

Notice that since Monte Carlo simulations introduce statistical noise, each pellet-wise Monte Carlo fission rate has an associated standard deviation. The RMS pellet-wise fission rate error of OpenMOC is 2.14% which is close to the average standard deviation in OpenMC tallies. The error in k_{eff} comes from ignoring the angular dependence of total cross-sections, leading to incorrect accounting of spatial self-shielding, particularly in U-238 [65, 66]. The radial and axial error distributions are presented in Figure 9-3 and Figure 9-4, respectively, showing a radial tilt from the center to the core periphery.

Figure 9-5 shows these assembly rate errors and the associated OpenMC reference folded into a 1/8 map. Similar to the radial distribution shown in Figure 9-3, the error again appears as a pure tilt. This tilt is due to the particular transport correction that is not able to fully capture the effects of anisotropic scattering. It is possible that a better transport correction – particularly in reflector regions – would be able to eliminate this tilt. However, accurate cross-section formation is not the objective of this thesis, and other ongoing projects at MIT are seeking to improve the transport correction across the core [66, 67]. The low number of inactive iterations (300) used in the Monte Carlo simulation could also be a significant contributor to the observed differences.

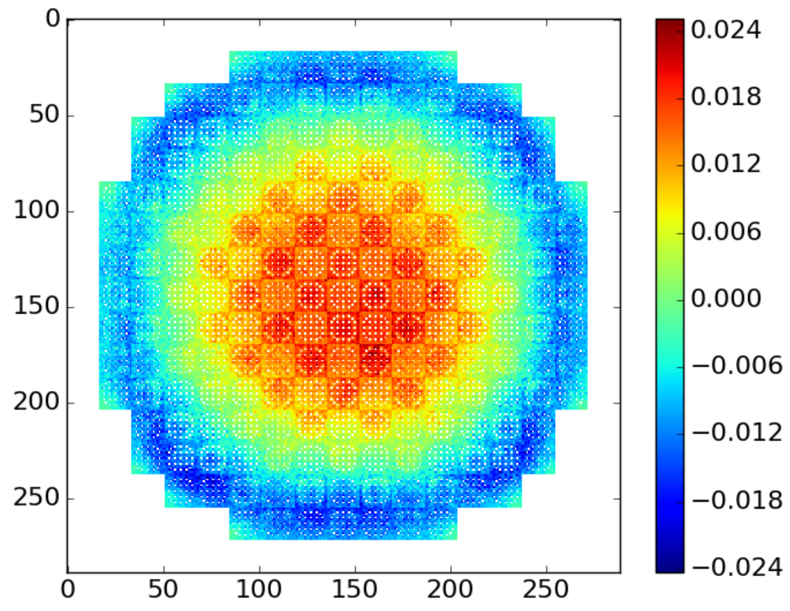


Figure 9-3: Radial distribution of normalized fission rate errors of OpenMOC compared with a reference OpenMC solution on the BEAVRS benchmark.

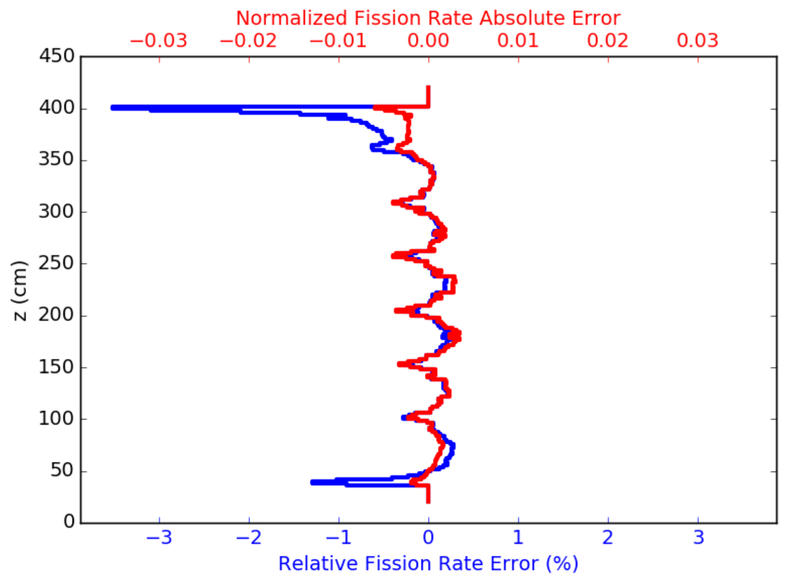


Figure 9-4: Axial error distribution of normalized fission rates of OpenMOC compared with a reference OpenMC solution on the BEAVRS benchmark.

9.2 Computational Performance

The computational requirements of the full core solution in OpenMOC is presented in Table 9.2. Note that while the number of core-hours required to converge the problem is

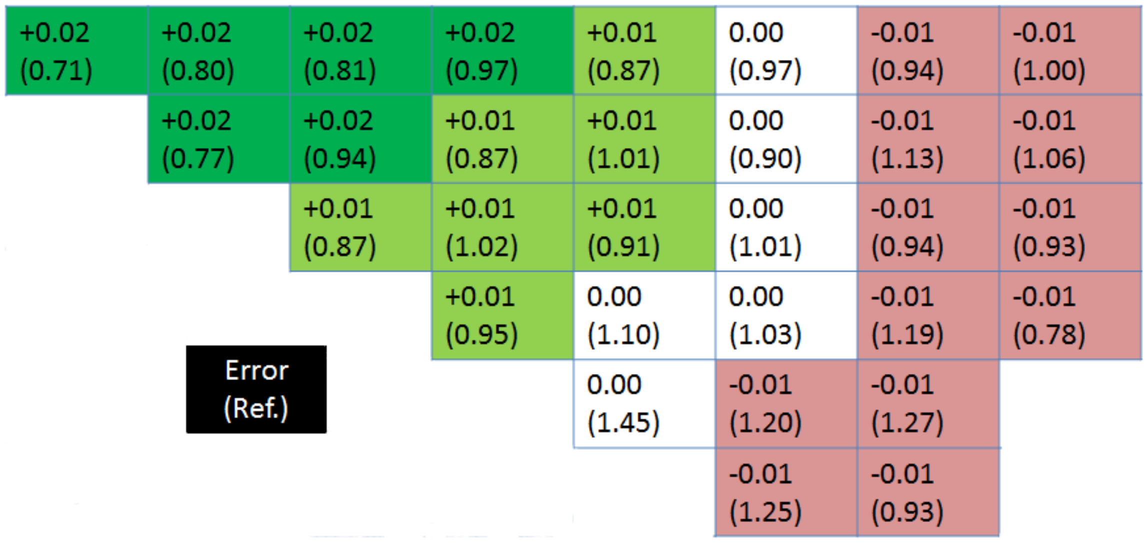


Figure 9-5: 1/8 core folded assembly fission rate error of OpenMOC compared with the reference OpenMC solution for the BEAVRS benchmark.

high (717,465), the solution was computed on the Argonne BlueGene/Q supercomputer, which has extremely slow cores for energy efficiency reasons. The requirements on modern computing cores, such as those on the Falcon supercomputer, is estimated by comparing the integration time of the SDSA problem tested in Chapter 5 on the Falcon supercomputer (47.9 ns) to those on the Argonne BlueGene/Q supercomputer (174.9 ns) for one node and all available cores. However, this case cannot be explicitly simulated due to the small aggregate memory of the Falcon supercomputer.

Table 9.2: Computational requirements of OpenMOC on the full core 3D BEAVRS benchmark using the Mira partition of the Argonne BlueGene/Q supercomputer

Runtime	7.76 hours
Number of Transport Sweeps	20
Number of Segments	3.44×10^{12}
Number of Source Regions	3.85×10^{8}
Nodes	5780 ($17 \times 17 \times 20$)
CPU Cores	92480
Integration Time	256.7 ns
Computational Cost	717,465 core-hours
Estimated Computational Cost on Falcon	$\approx$ 200,000 core-hours

Further analysis of the computational profile is presented in Table 9.3. The results show the computational overhead of CMFD acceleration was insignificant. In addition, the time spent communicating boundary angular fluxes between domains was quite small. However, there was significant idle time between sweeps when nodes are waiting for others to finish their current iteration. This imbalance is due to more work being required in core domains than in reflector domains.

Table 9.3: Computational profile of OpenMOC on the full core 3D BEAVRS benchmark using the Mira partition of the Argonne BlueGene/Q supercomputer

Solver Component			Computation Time (s)	Fraction of Runtime
Total			2.79×10^4	100%
	Transport Sweeps		2.67×10^4	95.8%
		Angular Flux Communication	7.05×10^2	2.5%
		Idle Time Between Sweeps	7.74×10^3	27.7%
	CMFD Solver		97.4	0.3%

9.3 Comparison with Flat Source MOC

Using the same MOC parameters, the BEAVRS core is simulated using the flat source solver. It is important to note that this case is not expected to be spatially converged since the mesh discretization was derived from linear source MOC sensitivity studies. Instead, the purpose is to compare the computational performance of the flat and linear source solvers to understand the overhead of the linear source approximation as well as possible differences in convergence behavior. The results are presented in Table 9.4, which again shows a factor of two to three overhead for the linear source solver. Also, the flat source solution converges 3 iterations faster than the linear source equivalent.

Table 9.4: Comparison of Flat and Linear source 3D MOC solvers on the BEAVRS benchmark with fixed mesh and ray spacing parameters

	Flat Source	Linear Source
Transport Sweeps	17	20
Run Time (hours)	3.13	7.76
Time per Iteration (minutes)	11.0	23.3
Core-hours	289,488	717,465
Integration Time (ns)	116.2	256.7

9.4 Parameter Refinement

In order to verify that the chosen MOC parameters were sufficiently converged in space and angle, simulations are conducted in which each 3D parameter is refined, with the results shown in Table 9.5. Ideally, all parameters should be refined together, but the computational burden would be too large to run on Mira with the allocation used in this study. Therefore, each parameter is refined separately. Due to memory constraints per domain, ray parameters were only able to be refined by $\approx 1.5\times$ but the axial mesh could be refined by a factor of 2.

Table 9.5: Differences observed from refining 3D MOC parameters for the BEAVRS benchmark relative to the first solution

Polar Angles	Axial Ray Spacing	Source Height	k_{eff} Bias	RMS Fission Rate Diff.	Max Fission Rate Diff.
10	0.75 cm	2.0 cm	–	–	–
14	0.75 cm	2.0 cm	2 pcm	0.010 (0.51%)	0.074 (1.92%)
10	0.50 cm	2.0 cm	5 pcm	0.008 (0.41%)	0.044 (1.76%)
10	0.75 cm	1.0 cm	13 pcm	0.001 (0.28%)	0.055 (3.33%)

9.5 Conclusions

In this Chapter, full core simulation results were presented for the BEAVRS benchmark. The results show reasonable agreement with the OpenMC Monte Carlo solution, though a radial tilt does exist, indicating either the need for a better transport correction or a better converged reference solution. Parameter refinement studies were conducted on the full core, showing some sensitivity to axial mesh. However, all parameter refinements nearly met the desired sensitivity criteria. The full core solution required 717,465 core-hours on the Argonne BlueGene/Q supercomputer with slow CPU cores. For Falcon CPU cores, representative of modern computer CPUs, it is estimated that only $\approx$ 200,000 core-hours would be required, and full core 3D MOC simulations are now feasible on modern supercomputers.

Chapter 10

Conclusions

The main goal of this thesis was to develop a 3D MOC solver capable of accurately and efficiently simulating LWR models for a fixed but reasonable choice of cross-sections. This goal was motivated by the desire to develop methods which can explicitly handle axial as well as radial variation within PWR reactor cores. The following summarizes the key accomplishments demonstrated by this thesis to meet this challenge:

- **The implementation of an efficient 3D MOC solver.** A 3D MOC solver was implemented in OpenMOC which is capable of efficiently solving the MOC equations using an efficient track laydown, on-the-fly ray tracing, and spatial domain decomposition. In addition, an efficient linear source solver was added, allowing for accurate solutions with relatively coarse mesh.

- **Theoretical and practical evaluation of source iteration convergence.** The convergence of transport codes using source iteration (such as MOC) with transport-corrected cross-sections has plagued researchers in the past. In this thesis, a robust theoretical framework is introduced to understand convergence characteristics. The diagonal stabilization scheme was presented which alleviates the convergence issues.

- **3D MOC parameter refinement studies.** Since previous 3D MOC solvers have not been capable of solving large PWR problems, parameter refinement studies have not been fully explored. In this thesis, the sensitivity of solution accuracy to each 3D MOC parameter is thoroughly explored.

- **Evaluation of computational requirements for solving full core PWR problems.** Since OpenMOC is the first solver capable of solving the BEAVRS benchmark with deterministic methods, it provides a useful indicator for the computational cost of solving such a large problem.

Past 3D deterministic solvers have not been capable of fully resolving full core PWR models to pellet-level precision. This thesis shows that these large scale simulations are now possible with careful consideration of implementation details critical to performance.

10.1 Summary of Work

10.1.1 3D MOC Implementation

In this thesis, **the track-based linear source approximation introduced by Ferrer** [42] **for 2D MOC is extended to 3D MOC** and implemented in OpenMOC. While the higher order source approximation adds a factor of $\approx 2 - 3\times$ computational cost for a fixed mesh, it also allows for a coarser mesh in the radial and axial directions while preserving solution accuracy.

The work in this thesis requires cyclic tracking in order to explicitly treat vacuum, periodic, and reflective boundary conditions. There are a variety of ways cyclic tracking can be enforced by adjusting MOC ray parameters. Depending on the track laydown algorithm, the parameter adjustment can be significant, often inserting far more tracks than necessary. Inefficient track laydown algorithms can lead to an order of magnitude increase in the number of tracks required for realistic PWR problems [18]. Since the computational cost of MOC scales directly with the number of tracks, an order of magnitude increase in the number of tracks translates directly into an order of magnitude increase in the run time. **This thesis implements track laydown using Modular Ray Tracing (MRT), which has less stringent constraints, allowing for the number of unnecessary track insertions to be minimal** [17].

Traditional MOC implementations conduct ray tracing upfront, storing the associated ray tracing data, and referencing it during the solve. While this approach is straightforward, its memory and compute requirements for 3D MOC are prohibitive, even for small problems, due to the vast number of segments present in 3D MOC simulations. The explicit storage of 3D segments in OpenMOC for a single assembly of the C5G7 benchmark with coarse MOC parameters required 79 GB [56]. Reducing the memory footprint is important for improving computational efficiency. In this thesis, **an alternative approach is presented that greatly reduces the segment storage by taking advantage of the extruded geometry structure common to many reactor physics problems**. This approach saves no 3D segment data, but instead stores 2D radial ray tracing information and combines this information with 1D axial meshes to compute 3D intersections on-the-fly [56].

Shared memory parallelism was implemented in OpenMOC for on-node parallelism. While shared memory parallelism is feasible for on-node scaling, it is infeasible for inter-node scaling due to significant latency when transferring information between nodes. Therefore, **a hybrid parallelism model is adopted in which on-node scaling is handled with OpenMP shared memory parallelism and inter-node scaling is handled with spatial domain decomposition using MPI** [14]. Scalable spatial domain decomposition was implemented in OpenMOC by partitioning the geometry using a uniform Cartesian grid into many geometrical sub-domains and imposing a modular track laydown [17] to naturally link tracks between sub-domains. The implementation shows nearly ideal weak scaling to a large number of nodes. CMFD acceleration was implemented in OpenMOC to reduce the number of transport sweeps necessary for convergence. This CMFD solver was also domain decomposed, such that its incorporation adds trivial overhead.

All of these implementation components in OpenMOC lead to an efficient 3D MOC solver, allowing for feasible large scale reactor physics simulations on modern supercomputers.

10.1.2 Diagonal Stabilization

While the implementation allows for efficient 3D MOC simulations, full core problems with transport-corrected cross-sections can be impossible to converge with conventional source iteration. **This thesis introduces a novel strategy to overcome the convergence issues of source iteration using damping of the MOC scalar fluxes, termed diagonal stabilization**. A robust description of the source of convergence issues with transport-corrected cross-sections was discussed in this thesis. Figure 10-1 illustrates the need for diagonal stabilization, depicting the convergence history of a full core BEAVRS simulation not converging with conventional source iteration but converging when diagonal stabilization is introduced.

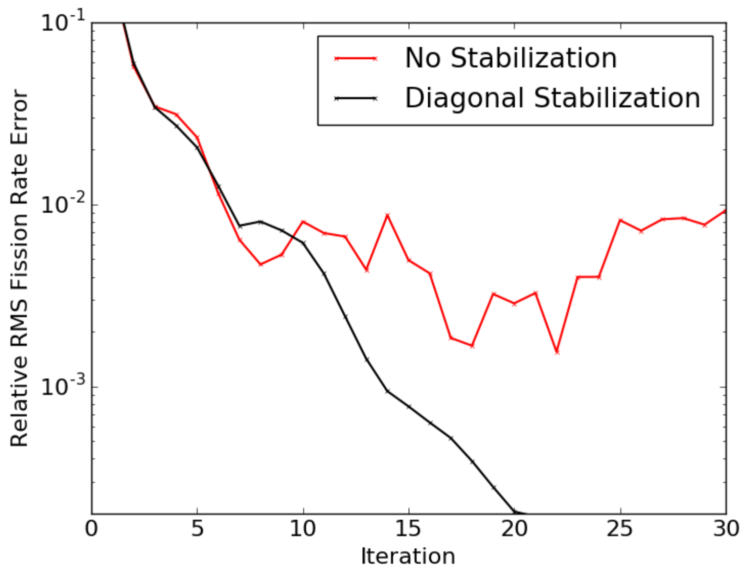


Figure 10-1: Convergence behavior of OpenMOC's linear source solver on the full core BEAVRS benchmark with and without diagonal stabilization.

Results show that for reactor problems with large water reflector regions and a high number of energy groups, convergence of MOC with reduced-group CMFD acceleration is not possible without a stabilization scheme, such as diagonal stabilization. Full group CMFD, in which the number of CMFD groups match the number of MOC groups, was always observed to converge. However, a reduced-group CMFD acceleration scheme is often desired in order to reduce overhead.

It is important to note that this stabilization strategy has implications for all neutron transport solvers – not just MOC solvers. Any solver which relies on source iteration has the potential to experience these same convergence issues when using transport-corrected cross-sections. Due to its wide-ranging applicability, this diagonal stabilization method is one of the most important contributions of this thesis to the reactor physics community.

10.1.3 Simulation Results

The OpenMOC 3D MOC solver was used to simulate a variety of problems formed from the BEAVRS benchmark. Using the parameters necessary to reach spatial and angular parameter convergence, the full core BEAVRS benchmark was simulated. The results are summarized in Table 10.1, showing reasonable agreement with a reference continuous energy OpenMC Monte Carlo solution, from which the multi-group cross-sections were derived.

Table 10.1: Simulation results of OpenMOC on the full core 3D BEAVRS benchmark using the Mira partition of the Argonne BlueGene/Q supercomputer compared with a reference OpenMC solution

OpenMOC k_{eff}	0.99677
OpenMC k_{eff}	0.99927 +/- 1×10^{-5}
OpenMOC RMS Fission Rate Error	2.14%
OpenMC Avg. Fission Rate Std. Dev.	1.82%
OpenMOC Runtime	7.76 hours
CPU Cores	92480
OpenMOC Computational Cost	717,465 core-hours
Estimated Computational Cost on Falcon	$\approx$ 200,000 core-hours

Note that while the number of core-hours required to converge the problem is high (717,465), the solution was computed on the Argonne BlueGene/Q supercomputer, which has extremely slow cores for energy efficiency reasons. The requirements on modern computing cores, such as those on the Falcon supercomputer was estimated to be 200,000 core-hours.

To verify the chosen parameters were sufficient in converging the spatial and angular effects, a perturbation study was conducted on the full core, showing little sensitivity when parameters were refined. A recap of the axial MOC parameters is given in Table 10.2.

Table 10.2: Axial MOC ray and mesh parameters determined to accurately and efficiently simulate the BEAVRS benchmark

Axial Ray Spacing	0.75 cm
Number of Polar Angles	10
Axial Source Height	2.0 cm

This thesis was able to accomplish the full core simulation of an LWR reactor, for the first time, using full core 3D deterministic neutron transport. These simulations can provide useful insight into the neutron behavior of rector geometries with complex geometric detail, such as the Westinghouse AP 1000™. With this insight, safety margins could potentially be lowered, leading to more efficient and economic operation of modern nuclear reactors.

10.2 Future Work

10.2.1 Accuracy Improvements of Full Core Simulations

The MOC solver developed in this thesis was able to reasonably simulate the BEAVRS benchmark. However, there was a noticeable tilt across the core due to the transport correction not properly accounting for anisotropic scattering. Therefore, this thesis illuminates the need for a better transport correction, particularly in reflector regions where the traditional flux-limited transport correction might not be sufficient. With an improved transport correction, the simulation results could be more accurate for the same computational cost.

10.2.2 Further Full Core Analysis

Future work in OpenMOC should also concentrate on reducing computational cost. Using the current solver, only uniform mesh refinement was studied for the axial direction in this thesis. A finer mesh could be used near reflector regions with a coarser mesh in the central core to improve accuracy and decrease computational cost. This would not require any further software development, only expanded analysis of full core reactor problems.

10.2.3 OpenMOC Improvements

Algorithmic implementation aspects of the OpenMOC could also be improved, including:

- **Non-uniform Domain Decomposition**: The requirement of uniform spatial domain decomposition leads to load balancing inefficiencies, as observed on the BEAVRS benchmark. If domains could be merged, this issue might be alleviated.

- **Reduced Boundary Angular Flux Storage**: OpenMOC currently requires double storage of boundary angular fluxes so that information is not overwritten during their exchange between nodes. However, it might be possible to only store the information once if a clever algorithm is implemented to prevent overwriting of information. Since the memory usage is dominated by boundary angular flux storage, this would reduce the overall memory requirements by nearly a factor of two. Other approaches could also be examined where no boundary fluxes are explicitly stored, as implemented in APOLLO3 [29] and ARRC [68].

- **Non-uniform CMFD Lattice**: Currently the OpenMOC CMFD implementation requires uniform mesh for CMFD acceleration. This is problematic for realistic full core problems where inter-assembly gaps exist. The inclusion of a uniform CMFD lattice inserts extra discretization into the problem as CMFD cell boundaries split MOC source regions.

- **Splitting of CMFD Cells Across Domain Boundaries**: Currently, domain boundaries cannot exist between CMFD cells. This is not ideal for the BEAVRS benchmark in which assemblies contain a 17×17 lattice of pins. Since pin-cell CMFD mesh is standard, this imposes limitations on how the assembly can be domain decomposed.

- **Inclusion of a GPU Solver**: For 2D MOC, Graphics Processing Units (GPUs) have shown the ability to solve MOC equations efficiently. Similar results should be possible for 3D MOC.

10.2.4 Spatial Source and Cross-section Approximations

In addition to specific OpenMOC improvements, other 3D MOC strategies could be introduced that have the potential for increased efficiency or accuracy using different spatial approximations.

First, different source approximations could also be studied in greater detail. Currently, a single source approximation (either flat or linear) is used for all regions in the core during a single simulation. However, within the modular framework, it is possible to create a solver which mixes flat and linear source approximations. For instance, moderator regions could always be simulated with a linear source approximation where there is a significant gradient, but gap and clad regions could be simulated with a flat source approximation where the neutron source is quite small. Additionally, source approximations restricted to only the axial direction, such as linear or quadratic, could be implemented for regions where radial variation is not significant.

In addition to implementing different source approximations, it would be useful to store spatial dependent cross-sections for depletion analysis. In current methodologies, fuel is discretized into many regions in order to account for burnup gradients. If the variation could be captured with a spatially-dependent cross-section approximation, coarser mesh could allow for decreased computational cost of depletion studies.

10.2.5 Treatment of Angular Dependence of Total Cross-sections

It was noted in Appendix F.2, the multi-group total cross-sections have an angular dependence and other authors have found a bias introduced by not accounting for the angular dependence [65]. Therefore, this should be treated in order to develop more accurate simulations capable of matching a fully converged continuous energy Monte Carlo solution. This could either be done by defining angular-dependent cross-sections, which could be computationally costly, or by correcting cross-sections through equivalence models [66].

10.2.6 Convergence of Source Iteration with Linear Sources and CMFD Acceleration

An important issue studied in this thesis was the convergence behavior of source iteration with transport-corrected cross-sections. However, the theoretical discussion of source iteration presented in this thesis relied on a flat source approximation without CMFD acceleration. The diagonal stabilization scheme was shown to also work for CMFD accelerated cases as well as the linear source solver, but a theoretical study would be useful, perhaps leading to an improved stabilization strategy.

10.2.7 Reducing the Computational Requirements of Full Core Simulations

Finally, the development of 3D transport methods should focus on making high fidelity reactor simulations even more efficient. The results presented in this thesis used many-group cross-section libraries and solution of the BEAVRS benchmark required a large supercomputer. These many-group cross-section libraries were used to reduce the spatial variation of cross-sections such that they are only dependent on the material, not the spatial location. However, LWR simulations would be far more feasible if region-dependent several-group cross-sections were capable of accurately capturing neutron behavior. Therefore, future analysis should investigate several-group cross-section formations which maintain solution accuracy.

Appendices

Appendix A

Matrix Representation of MOC

This appendix focuses on how to solve the MOC system of equations, focusing on the flat source approximation, as the equations are far simpler. However, a linear source approximation would lead to a similar discussion. In Chapter 2, Eq. 2.24 and Eq. 2.27 provided ways to calculate the angular fluxes and scalar fluxes, respectively. The source can be computed from the scalar fluxes with Eq. 2.26. This forms a system of equations that can be solved to determine the neutron distribution inside a nuclear reactor core.

In this discussion, the problem will be cast as a set of matrix equations that could theoretically be solved with common linear algebra packages. However, this discussion will show that blindly solving the system of equations with a general linear algebra package is computationally infeasible since it loses sight of inherent structure the physics-based approach naturally captures.

The system of equations to be solved is formed by Eq. 2.26, Eq. 2.24, and Eq. 2.27. Turning Eq. 2.26 into matrix form, a fission matrix F and a scattering matrix S are defined such that

$$\mathbf{q} = \frac{1}{k} F \boldsymbol{\phi} + S \boldsymbol{\phi} \tag{A.1}$$

where $\mathbf{q}$ is a vector of size M containing all neutron sources and $\boldsymbol{\phi}$ is a vector of size M containing all scalar fluxes. Since there must be a neutron source and scalar flux for every source region and every energy group, $M = LG$ where L is the number of source regions and G is the number of groups. The scalar fluxes $\boldsymbol{\phi}$ as well as the sources $\mathbf{q}$

are ordered such that they are contiguous in group with the index calculated as $iG+g$. In the context of this discussion, elements relating to region quantities are indexed by source region i and group g, yielding the following definitions for the fission matrix F and scattering matrix S as

$$F_{(i,g),(i,g')} = \frac{1}{4\pi}\chi_{i,g}\,\nu\Sigma_f^{i,g'} \tag{A.2}$$

and

$$S_{(i,g),(i,g')} = \frac{1}{4\pi}\Sigma_s^{i,g'\to g} \tag{A.3}$$

where all other unspecified matrix elements are zero. Both of these matrices are of size $M \times M$ and sparse since there are no inter-regional terms.

The relationship in Eq. 2.24 can be rearranged to form the relationship in Eq. A.4. This form is much easier to work with in the translation to matrix definitions.

$$\psi_g^{t,\varsigma}(0)\left(\frac{F_1\left(\Sigma_t^{i,g}s\right)-1}{F_1\left(\Sigma_t^{i,g}s\right)}\right)+\psi_g^{t,\varsigma}(s)\left(\frac{1}{F_1\left(\Sigma_t^{i,g}s\right)}\right)=\frac{q_{i,g}^0}{\Sigma_t^{i,g}} \tag{A.4}$$

This relationship can be turned into matrix form by defining an angular flux vector $\boldsymbol{\psi}$ that contains all outgoing angular fluxes. This is represented in Eq. A.5 by an angular flux transport matrix T defining relationships between angular fluxes, a source selection matrix H which selects the source of the region being traversed, and a diagonal matrix U containing the total cross-sections which scale the source appropriately to match the relationship in Eq. A.4.

$$T\boldsymbol{\psi} = HU^{-1}\mathbf{q} \tag{A.5}$$

The number of angular fluxes is $N = \beta LG$ where β is the average number of track crossings per source region. Since there must be a significant number of track crossings per region for convergence we expect $N >> M$. T is size $N \times N$ as it defines relationships between angular fluxes, the size of H is $N \times M$ since for each angular flux pair it must pick out the appropriate source region, and the size of U is $M \times M$ since it relates only to the source regions. The elements relating to angular flux quantities are indexed by track t, segment ς, and group g. For notational convenience, a function R is created which maps a track t and segment number ς to the traversed region $R(t,\varsigma)$. The source selection matrix H can therefore be defined as

$$H_{(t,\varsigma,g),(R(t,\varsigma),g)} = 1. \tag{A.6}$$

This matrix therefore, has only one non-zero value per row, indicating which region is being traversed, relating track-based quantities such as angular fluxes to region based quantities such as the scalar fluxes. Its transpose similarly relates the regions to the tracks traversing the region. The matrix $H^T H$ is a $M \times M$ diagonal matrix with each diagonal element representing the number of tracks that traverse the region multiplied by the number of groups. Since it is diagonal, it is easily invertible, which will be important in the later discussion. The diagonal matrix U containing the total cross-sections is defined by

$$U_{(i,g),(i,g)} = \Sigma_t^{i,g}. \tag{A.7}$$

The angular flux transport matrix T is defined by

$$T_{(t,\varsigma,g),(t,\varsigma,g)} = \frac{1}{F_1\left(\Sigma_t^{R(t,\varsigma),g} \ell_{t,\varsigma}\right)} \tag{A.8}$$

and

$$T_{(t,\varsigma,g),(t,\varsigma-1,g)} = \frac{F_1\left(\Sigma_t^{R(t,\varsigma),g} \ell_{t,\varsigma}\right) - 1}{F_1\left(\Sigma_t^{R(t,\varsigma),g} \ell_{t,\varsigma}\right)}. \tag{A.9}$$

Again, all non-specified quantities are zero.

Lastly, Eq. 2.27 describes how the scalar flux is calculated in terms of both a weighted sum of angular fluxes and the neutron source. Specifically, an angular flux weighting matrix W of size $M \times N$ is defined such that

$$\boldsymbol{\phi} = U^{-1}\mathbf{q} + U^{-1} W \boldsymbol{\psi}. \tag{A.10}$$

In order to expose its structure, W is expressed as a multiplication of matrices

$$W = V^{-1} H^T \tilde{W} \tag{A.11}$$

where the volume matrix V is a $M \times M$ diagonal matrix containing region volumes as

$$V_{(i,g),(i,g)} = V_i \tag{A.12}$$

and $\tilde{W}$ is an $N \times N$ matrix defined by

$$\tilde{W}_{(t,\varsigma,g),(t,\varsigma,g)} = -w_t \tag{A.13}$$

and

$$\tilde{W}_{(t,\varsigma,g),(t,\varsigma-1,g)} = w_t \tag{A.14}$$

with all other elements being zero. Now that all matrix elements have been defined, the transport equation can be written as a matrix eigenvalue problem. Combining Eq. A.1 and Eq. A.10 yields

$$\boldsymbol{\phi} = U^{-1}\left(\frac{1}{k}F + S\right)\boldsymbol{\phi} + U^{-1}W\boldsymbol{\psi} \tag{A.15}$$

which combined with Eq. A.5 yields

$$\boldsymbol{\phi} = U^{-1}\left(I + WT^{-1}HU^{-1}\right)\left(\frac{1}{k}F + S\right)\boldsymbol{\phi} \tag{A.16}$$

where I is the identity matrix of dimension $M \times M$. This relationship can be further simplified by defining a matrix J such that

$$J = U^{-1}\left(I + WT^{-1}HU^{-1}\right) \tag{A.17}$$

leading to the expression

$$\boldsymbol{\phi} = J\left(\frac{1}{k}F + S\right)\boldsymbol{\phi} \tag{A.18}$$

This results in the generalized eigenvalue equation given in Eq. A.19

$$A\boldsymbol{\phi} = kB\boldsymbol{\phi} \tag{A.19}$$

where

$$A = JF \tag{A.20}$$

and

$$B = I - JS. \tag{A.21}$$

This can of course be turned into a regular eigenvalue problem by explicitly taking the inverse as

$$B^{-1}A\boldsymbol{\phi} = k\boldsymbol{\phi}. \tag{A.22}$$

At this point, the matrix $B^{-1}A$ could be explicitly calculated and then input into any standard eigenvalue solver. However, taking matrix inverses is very computationally intense, especially due to the internal structure of A and B. Specifically, since J involves the inverse of T, even the explicit computation of its elements is infeasible. Therefore, doing this would be very unwise. Even if a generalized eigenvalue solver is available capable of solving equations of the form given in Eq. A.19, the problem would still rely on computing explicit components of J. Even though the steady-state neutron transport equation is an eigenvalue problem defined in terms of the angular fluxes, they only enter the equation implicitly with the inversion of T.

Therefore, instead of using a common eigenvalue solution technique, a variation of fixed point iteration termed *source iteration* is chosen to solve the system. In this procedure, the relationship in Eq. A.16 is used with the right hand side of the equation lagged as

$$\boldsymbol{\phi}_{n+1} = U^{-1}\left(I + WT^{-1}HU^{-1}\right)\left(\frac{1}{k_n}F + S\right)\boldsymbol{\phi}_n \tag{A.23}$$

where the subscript n indicates iteration number. Mechanically, the process iterates over estimations of the neutron source, calculating the corresponding fluxes. First, an initial flux distribution is guessed along with a value for the eigenvalue k. At the start of each iteration, the source distribution $\mathbf{q}$ is calculated using Eq. A.1 from the current guess of scalar fluxes $\boldsymbol{\phi}$ and eigenvalue k. Then, during the *transport sweep*, new angular fluxes are computed as

$$\boldsymbol{\psi} = T^{-1} H U^{-1} \mathbf{q}. \tag{A.24}$$

First, the computation of $HU^{-1}\mathbf{q}$ is trivial since H is just the source selection matrix and U is diagonal. Physically, this just relates to selecting the source region being traversed and calculating the source divided by the total cross-section.

Due to the simple structure of T, solving its *implicit* inversion is rather simple for a matrix of its size. T has just one element on the diagonal and one off-diagonal element per row. However, since the size of T is so large, any operation involving the matrix is still expensive. Note that the angular fluxes are only related to their associated connecting angular flux. For some boundary conditions (e.g. reflective), the connecting angular flux might be from another track. This could cause the inversion to be more difficult. If all boundaries have this characteristic, all the angular fluxes within a cycle of connecting tracks would be dependent on each other.

To alleviate this issue, the angular fluxes at boundaries are approximated by the calculated angular flux at the boundary from the previous iteration. Specifically, the relationship in Eq. 2.11 relating connecting angular fluxes at the start of a track is approximated by

$$\psi_g^{t,1}(0) = \widetilde{\psi}_g^{C(t),S(C(t))}(\ell_{C(t),S(C(t))}) \tag{A.25}$$

where $\widetilde{\psi}$ represents the calculated angular fluxes from the previous iteration so that $\widetilde{\psi}_g^{C(t),S(C(t))}$ is the angular flux of the connecting track from the previous iteration. This transformation allows the inversion of T to be calculated using an altered matrix $\tilde{T}$ which lacks dependency between different tracks, then adding the contribution of the previous iteration angular fluxes, if applicable. The structure of $\tilde{T}$ is block-diagonal and within each block the matrix is upper triangular. Physically this means that all tracks can be calculated independently of each other during an iteration. For each track, the angular fluxes can be solved sequentially by segment. This is where the *transport sweep* owes its name as the algorithm simply sweeps over segments. Since each row has at most two elements, very little calculation is required for each angular flux.

During this process, it is noted that only the boundary angular fluxes along with the neutron source are needed to determine all angular fluxes. Therefore, non-boundary angular fluxes are computed on-the-fly. Furthermore, the elements of T are re-computed on-the-fly. Once an angular flux is computed, its contribution to the scalar fluxes is tallied before it is discarded, since the computation of the scalar fluxes $\boldsymbol{\phi}$ relies on a weighted sum of the full angular flux vector $\boldsymbol{\psi}$.

After the transport sweep, the source is added to the scalar flux tally, consistent with Eq. A.10, to produce a new estimate of the scalar fluxes $\boldsymbol{\phi}$. To form a new estimate of k, note that

$$T\boldsymbol{\psi} = \frac{1}{k}HU^{-1}F\boldsymbol{\phi} + HU^{-1}S\boldsymbol{\phi} \tag{A.26}$$

which then is re-arranged and multiplied with a vector of ones $\mathbb{1}_M$ of length M as

$$\mathbb{1}_M^T UH_{\text{left}}^{-1}T\boldsymbol{\psi} = \frac{1}{k}\mathbb{1}_M^T F\boldsymbol{\phi} + \mathbb{1}_M^T S\boldsymbol{\phi}. \tag{A.27}$$

Since H is rectangular, it cannot be simply inverted. But since it is of full rank, $H^T H$ can be inverted. From the previous discussion of H, recall that $H^T H$ is diagonal so its inversion is simple. Therefore the left inverse H_{left}^{-1} of matrix H is defined to be

$$H_{\text{left}}^{-1} = \left(H^T H\right)^{-1} H^T. \tag{A.28}$$

This allows the eigenvalue k to be computed as

$$k = \frac{\mathbb{1}_M^T F\boldsymbol{\phi}}{\mathbb{1}_M^T UH_{\text{left}}^{-1}T\boldsymbol{\psi} - \mathbb{1}_M^T S\boldsymbol{\phi}}. \tag{A.29}$$

Combining Eq. A.5 and Eq. A.10, note that

$$UH_{\text{left}}^{-1}T\boldsymbol{\psi} = U\boldsymbol{\phi} - W\boldsymbol{\psi} \tag{A.30}$$

and therefore

$$k = \frac{\mathbb{1}_M^T F\boldsymbol{\phi}}{\mathbb{1}_M^T (U-S)\,\boldsymbol{\phi} - \mathbb{1}_M^T W\boldsymbol{\psi}}. \tag{A.31}$$

where $\mathbb{1}_M^T F \boldsymbol{\phi}$ refers to the fission rate, $\mathbb{1}_M^T (U - S) \boldsymbol{\phi}$ refers to the absorption rate, and $-\mathbb{1}_M^T W \boldsymbol{\psi}$ refers to the leakage rate. Physically, this shows that k is simply a ratio of neutron production to neutron loss terms.

Notice that each source iteration relies on simply taking the inverse of the transport matrix T rather than the full B matrix, which includes the scattering matrix, as would normally be taken during an eigenvalue solver, e.g. power iteration. While this choice does ease the computational burden of each iteration, it also requires many more iterations to converge to the correct solution as each iteration does little work in resolving the angular fluxes as the scattering matrix is not included in the inversion.

Physically, this relates to not treating the scattering source as directly coupled to the total neutron source. Therefore, un-accelerated MOC is inherently plagued by slow convergence. In OpenMOC, CMFD acceleration is used to resolve the issue of slow convergence for source iteration, as described in Appendix B. This allows for the computational burden of each iteration to be eased without giving up anything in terms of convergence rate by iteration.

Appendix B

CMFD Acceleration

Often CMFD acceleration is necessary to achieve reasonable convergence on practical reactor physics problems [69]. This appendix explains CMFD acceleration, as implemented in OpenMOC. During CMFD acceleration, a coarse mesh problem is solved that is consistent with the fine mesh MOC problem. Since the coarse mesh problem can be solved quickly, it can be fully converged and used to update the MOC solution allowing global behavior to be communicated in fewer iterations. This appendix discuss CMFD as a multigrid method, shows how the CMFD equations are derived from the fundamental multi-group transport equation, and then the process of applying CMFD acceleration is discussed.

B.1 Multigrid Methods

Multigrid methods are popular in numerical analysis for solving differential equations. The fundamental idea is that global information can be transferred much quicker over a coarse mesh than a fine mesh. Using this principle multigrid methods alternate between solving the set of equations on a coarse mesh, where the problem size is reduced and information propagates much quicker, and a fine mesh where the discretization accurately captures the solution of the problem. It is vital that the coarse mesh solution be consistent with the fine mesh solution. In this context, consistency means that at convergence the coarse mesh and fine mesh solutions agree over the coarse mesh.

Multigrid methods can be structured in many different ways but they generally involve two important stages: restriction and prolongation.

- **Restriction -** Collapsing the fine mesh solution down to a consistent form on a coarse mesh.
- **Prolongation -** Using the coarse mesh solution to interpolate corrections to the fine mesh solution.

Multigrid methods in general can involve many layers of mesh. At each layer, restriction and prolongation are used to transfer to coarser and finer mesh layers, respectively. However, for the CMFD acceleration scheme implemented in this thesis, only two layers of mesh are used: the fine MOC mesh and the coarse CMFD mesh. Figure B-1 illustrates the process of solving the MOC equations using CMFD acceleration.

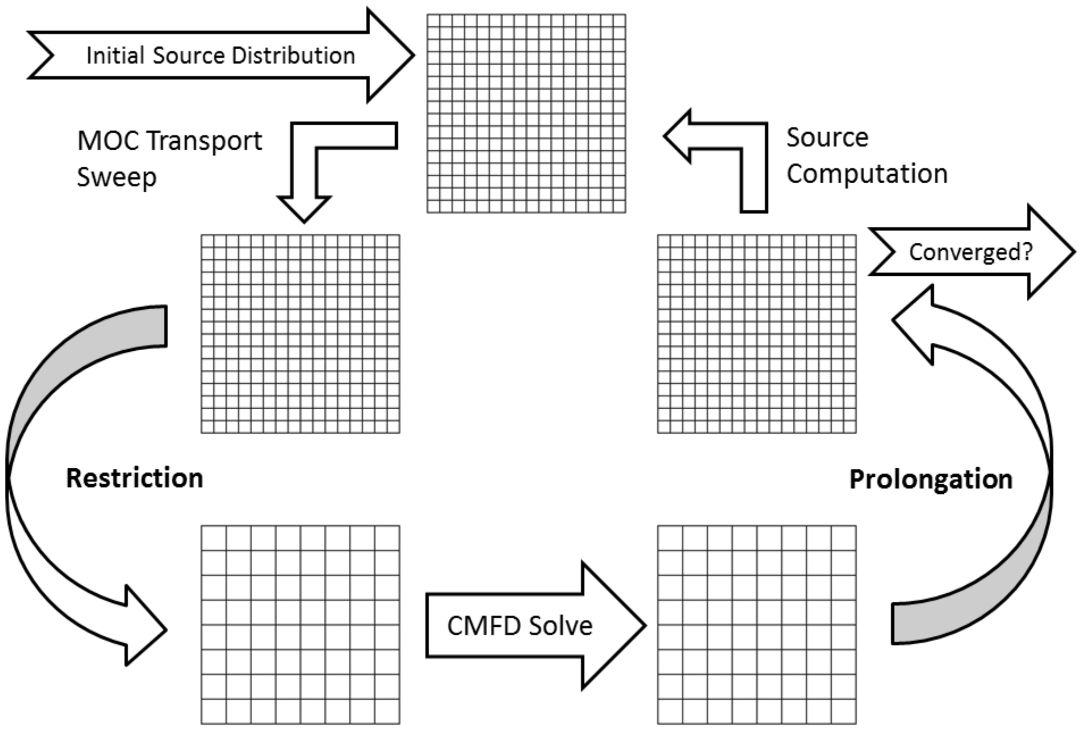


Figure B-1: A depiction of the multigrid approach to solving the MOC transport equations with CMFD acceleration.

One important difference between usual multigrid approaches and CMFD is that the CMFD equations are solved using consistent but fundamentally different equations. Instead of solving the coarse mesh problem using the same MOC form of the neutron transport equation, where the angular space is discretized using tracks, the CMFD equations rely on a diffusion-like formulation. This alternate form of the transport equation relies strictly on cell averaged quantities rather than having a dependence on angular directions.

B.2 Derivation of the CMFD Equations

The CMFD equations can be derived from multi-group transport. The general concept is to turn the transport equation, which is fundamentally based on angular fluxes into a diffusion-like problem which is fundamentally based on scalar fluxes averaged over some volume in the reactor. During this process, some approximations will be introduced. However, all of these approximations introduce no bias at convergence. Therefore they do not impact solution accuracy. First recall the multigroup approximation given in Eq. 2.4:

$$\mathbf{\Omega}\cdot\nabla\psi_g(\mathbf{r},\mathbf{\Omega})+\Sigma_t^g(\mathbf{r})\psi_g(\mathbf{r},\mathbf{\Omega}) = \frac{1}{4\pi}\left(\frac{\chi_g(\mathbf{r})}{k}\sum_{g'=1}^{G}\nu_{g'}(\mathbf{r})\,\Sigma_f^{g'}(\mathbf{r})\,\phi_{g'}(\mathbf{r}) + \sum_{g'=1}^{G}\Sigma_s^{g'\to g}(\mathbf{r})\,\phi_{g'}(\mathbf{r})\right) \tag{B.1}$$

To transform this equation into one based on the scalar fluxes $\phi_g(\mathbf{r})$ rather than the angular fluxes $\psi_g(\mathbf{r},\mathbf{\Omega})$, the equation is integrated over the entire 4π angular space. In doing so, recall from Eq. 2.2 that the integral of angular flux over the entire angular space is simply the scalar flux, leading to the relationship in Eq. B.2.

$$\int_{4\pi} d\mathbf{\Omega}\,\mathbf{\Omega}\cdot\nabla\psi_g(\mathbf{r},\mathbf{\Omega})+\Sigma_t^g(\mathbf{r})\,\phi_g(\mathbf{r}) = \frac{\chi_g(\mathbf{r})}{k}\sum_{g'=1}^{G}\nu_{g'}(\mathbf{r})\,\Sigma_f^{g'}(\mathbf{r})\,\phi_{g'}(\mathbf{r})+\sum_{g'=1}^{G}\Sigma_s^{g'\to g}(\mathbf{r})\,\phi_{g'}(\mathbf{r}) \tag{B.2}$$

Notice that only the streaming term has any dependence on the angular flux $\psi_g(\mathbf{r},\mathbf{\Omega})$. Since the angular variable $\mathbf{\Omega}$ is independent of the spatial variable $\mathbf{r}$, the gradient can be brought outside the integral in the streaming term as shown in Eq. B.3.

$$\nabla\cdot\int_{4\pi} d\mathbf{\Omega}\,\mathbf{\Omega}\psi_g(\mathbf{r},\mathbf{\Omega})+\Sigma_t^g(\mathbf{r})\,\phi_g(\mathbf{r})=\frac{\chi_g(\mathbf{r})}{k}\sum_{g'=1}^{G}\nu_{g'}(\mathbf{r})\,\Sigma_f^{g'}(\mathbf{r})\,\phi_{g'}(\mathbf{r})+\sum_{g'=1}^{G}\Sigma_s^{g'\to g}(\mathbf{r})\,\phi_{g'}(\mathbf{r}) \tag{B.3}$$

Next, the net current $J_g(\mathbf{r})$ is defined as

$$J_g(\mathbf{r})=\int_{4\pi} d\mathbf{\Omega}\,\mathbf{\Omega}\psi_g(\mathbf{r},\mathbf{\Omega}). \tag{B.4}$$

Inserting this definition and integrating the equation over an arbitrary volume V leads to Eq. B.5.

$$\begin{aligned}\int_V d\mathbf{r}\,\nabla\cdot J_g(\mathbf{r})+\int_V d\mathbf{r}\,\Sigma_t^g(\mathbf{r})\,\phi_g(\mathbf{r})=&\\ \int_V d\mathbf{r}\left(\frac{\chi_g(\mathbf{r})}{k}\sum_{g'=1}^{G}\nu_{g'}(\mathbf{r})\,\Sigma_f^{g'}(\mathbf{r})\,\phi_{g'}(\mathbf{r})+\sum_{g'=1}^{G}\Sigma_s^{g'\to g}(\mathbf{r})\,\phi_{g'}(\mathbf{r})\right)&\end{aligned} \tag{B.5}$$

The entire geometry is then partitioned into CMFD cells. Defining a volume V_j for CMFD cell j which is the composition of a finite number of non-overlapping MOC source regions, the transport equation can be cast in terms of the fluxes and constant cross-sections for the MOC source regions i with volumes V_i as given in Eq. B.6.

$$\begin{aligned}\int_{V_j} d\mathbf{r}\,\nabla\cdot J_g(\mathbf{r})+\sum_{i\in j}\int_{V_i} d\mathbf{r}\,\Sigma_t^{i,g}\phi_g(\mathbf{r})=&\\ \sum_{i\in j}\int_{V_i} d\mathbf{r}\left(\frac{\chi_{i,g}}{k}\sum_{g'=1}^{G}\nu_{i,g'}\Sigma_f^{i,g'}\phi_{g'}(\mathbf{r})+\sum_{g'=1}^{G}\Sigma_s^{i,g'\to g}\phi_{g'}(\mathbf{r})\right)&\end{aligned} \tag{B.6}$$

It is important to note that in this definition, CMFD boundaries are not allowed to intersect MOC source region boundaries. In practice, source regions with an intersecting CMFD boundary are split so that each source region has only one CMFD cell within its domain. Since the MOC equations are often solved for the average flux within source regions, $\overline{\phi_{i,g}}$, the transport equation can be re-written as:

$$\int_{V_j} d\mathbf{r}\, \nabla \cdot J_g(\mathbf{r}) + \sum_{i\in j} \Sigma_t^{i,g} \overline{\phi_{i,g}} V_i = \sum_{i\in j} \left(\frac{\chi_{i,g}}{k} \sum_{g'=1}^{G} \nu_{i,g'} \Sigma_f^{i,g'} \overline{\phi_{i,g'}} V_i + \sum_{g'=1}^{G} \Sigma_s^{i,g'\to g} \overline{\phi_{i,g'}} V_i \right) \tag{B.7}$$

All of the variables given in this equation are present in the MOC equations except for the net current found in the streaming term. The bounding surface of CMFD cell j is defined as S_j with surface normal $\mathbf{n}$. Applying gauss-divergence theorem to the streaming term, it can be cast as a surface integral, shown in Eq. B.8.

$$\int_{S\in S_j} dS\, J_g(\mathbf{r}) \cdot \mathbf{n} + \sum_{i\in j} \Sigma_t^{i,g} \overline{\phi_{i,g}} V_i = \sum_{i\in j} \left(\frac{\chi_{i,g}}{k} \sum_{g'=1}^{G} \nu_{i,g'} \Sigma_f^{i,g'} \overline{\phi_{i,g'}} V_i + \sum_{g'=1}^{G} \Sigma_s^{i,g'\to g} \overline{\phi_{i,g'}} V_i \right) \tag{B.8}$$

In order to make the CMFD problem even less computationally intense, the group structure is coarsened. A group collapse is performed in which a given CMFD group e is formed from the incorporation of one or more MOC groups. To arrive at this relationship, the transport equation is summed over all MOC groups g within the CMFD group e, shown in Eq. B.9.

$$\begin{gathered} \sum_{g\in e} \left(\int_{S\in S_j} dS\, J_g(\mathbf{r}) \cdot \mathbf{n} + \sum_{i\in j} \Sigma_t^{i,g} \overline{\phi_{i,g}} V_i \right) = \\ \sum_{i\in j} \left(\frac{\sum_{g\in e} \chi_{i,g}}{k} \sum_{g'=1}^{G} \nu_{i,g'} \Sigma_f^{i,g'} \overline{\phi_{i,g'}} V_i + \sum_{g'=1}^{G} \left(\sum_{g\in e} \Sigma_s^{i,g'\to g} \right) \overline{\phi_{i,g'}} V_i \right) \end{gathered} \tag{B.9}$$

CMFD cross-sections over coarse mesh and coarse group structures are defined in terms of the fine mesh MOC quantities. The CMFD cross-sections are given the subscript C and their definitions are given in equations B.10 – B.13.

$$\chi_C^{j,e} = \frac{\sum_{i\in j} \left[\left(\sum_{g\in e} \chi_{i,g} \right) \left(\sum_{g'=1}^{G} \nu_{i,g'} \Sigma_f^{i,g'} \overline{\phi_{i,g'}} V_i \right) \right]}{\sum_{i\in j} \sum_{g=1}^{G} \nu_{i,g} \Sigma_f^{i,g} \overline{\phi_{i,g}} V_i} \tag{B.10}$$

$$\nu_C^{j,e}\Sigma_{C,f}^{j,e} = \frac{\sum_{i\in j}\sum_{g\in e}\nu_{i,g}\Sigma_f^{i,g}\overline{\phi_{i,g}}V_i}{\sum_{i\in j}\sum_{g\in e}\overline{\phi_{i,g}}V_i} \tag{B.11}$$

$$\Sigma_{C,s}^{j,e'\rightarrow e} = \frac{\sum_{i\in j}\sum_{g'\in e'}\left(\sum_{g\in e}\Sigma_s^{i,g'\rightarrow g}\right)\overline{\phi_{i,g'}}V_i}{\sum_{i\in j}\sum_{g\in e}\overline{\phi_{i,g}}V_i} \tag{B.12}$$

$$\Sigma_{C,t}^{j,e} = \frac{\sum_{i\in j}\sum_{g\in e}\Sigma_t^{i,g}\overline{\phi_{i,g}}V_i}{\sum_{i\in j}\sum_{g\in e}\overline{\phi_{i,g}}V_i} \tag{B.13}$$

Notice that these cross-sections involve a weighting over the MOC fluxes. Therefore the CMFD solution depends on the MOC scalar fluxes. At convergence, there is no approximation error, allowing the CMFD solution to be entirely consistent. CMFD cell volumes V_C^j and cell-averaged scalar fluxes $\phi_C^{j,e}$ are defined by Eq. B.14 and Eq. B.15, respectively.

$$V_C^j = \sum_{i\in j} V_i \tag{B.14}$$

$$\phi_C^{j,e} = \frac{\sum_{i\in j}\sum_{g\in e}\overline{\phi_{i,g}}V_i}{\sum_{i\in j}V_i} \tag{B.15}$$

These definitions, along with the CMFD cross-section definitions, form the basis of the restriction component of CMFD acceleration. It is important to note that the CMFD mesh is significantly coarser - both in space and energy – than the MOC mesh. A comparison of the spatial mesh is given in Figure B-2 with the MOC mesh on the left. The depicted mesh are the actual mesh sizes used in the final results for this thesis. The MOC mesh is quite coarser than typical MOC mesh due to the use of a linear source approximation. The CMFD calculations in this thesis use pin-cell sized mesh. Even with the coarse MOC mesh, the CMFD mesh is significantly coarser.

For the energy condensation, the MOC calculations in this thesis use 70 energy groups whereas the CMFD solver uses 25 or less energy groups. The combination of coarse spatial mesh and coarse energy groups causes the CMFD problem size to be incredibly small in comparison with the MOC problem size. With the collapsed cross-sections on the coarse mesh, the CMFD transport equation looks very similar to the original

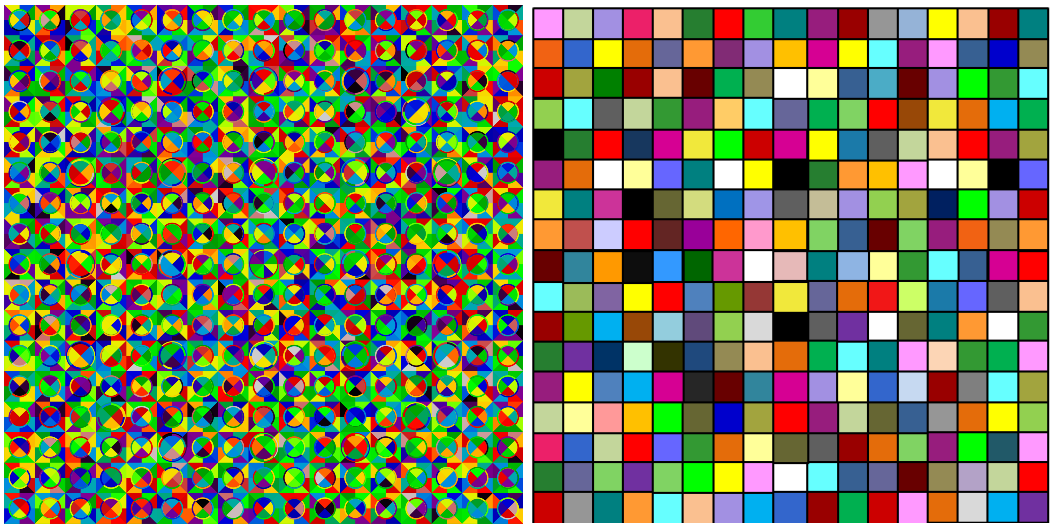

(a) **(b)**

Figure B-2: A depiction of the spatial mesh used in MOC (a) and CMFD (b) solvers. The mesh refinements correspond to those used in the final results for this thesis.

multi-group transport equation and is given in Eq. B.16.

$$\frac{1}{V_C^j}\sum_{g\in e}\left(\int\limits_{S\in S_j} dS\, J_g(\mathbf{r})\cdot\mathbf{n}\right) + \Sigma_{C,t}^{j,e}\phi_C^{j,e} = \frac{\chi_C^{j,e}}{k}\sum_{e'=1}^{E}\nu_C^{j,e'}\Sigma_{C,f}^{j,e'}\phi_C^{j,e'} + \sum_{e'=1}^{E}\Sigma_{C,s}^{i,e'\to e}\phi_C^{j,e'} \tag{B.16}$$

Returning to the streaming term, the entire surface S_j of CMFD cell j is partitioned into H surfaces that form an interface between cell j and exactly one other CMFD cell. This allows the total net current of CMFD cell j to be defined in terms of the sum over net currents over these interfacial surfaces as given in Eq. B.17.

$$\frac{1}{V_C^j}\sum_{g\in e}\sum_{h=1}^{H}\left(\int\limits_{S\in S_{j,h}} dS\, J_g(\mathbf{r})\cdot\mathbf{n}\right) + \Sigma_{C,t}^{j,e}\phi_C^{j,e} = \frac{\chi_C^{j,e}}{k}\sum_{e'=1}^{E}\nu_C^{j,e'}\Sigma_{C,f}^{j,e'}\phi_C^{j,e'} + \sum_{e'=1}^{E}\Sigma_{C,s}^{i,e'\to e}\phi_C^{j,e'} \tag{B.17}$$

The integrated net current over each interfacial surface for an MOC group g can be cast in terms of angular fluxes using the definition given in Eq. B.4 to produce the relationship in Eq. B.18.

$$\int\limits_{S_{j,h}} dS\, J_g(\mathbf{r})\cdot\mathbf{n} = \int\limits_{S\in S_{j,h}} dS \int\limits_{4\pi} d\mathbf{\Omega}\,\psi_g(\mathbf{r},\mathbf{\Omega})\,(\mathbf{\Omega}\cdot\mathbf{n}) \tag{B.18}$$

Figure B-3 shows the geometric relationship between a given MOC track and CMFD surfaces. This shows that the surface area penetrated on surface $S_{j,h}$ by track t can be calculated as $\delta A_t/(\mathbf{\Omega}\cdot\mathbf{n})$. With this geometric relationship in mind and factoring in

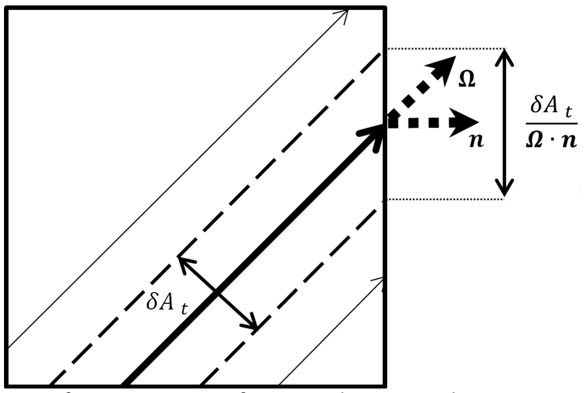


Figure B-3: A depiction of a CMFD surface with normal vector $\mathbf{n}$ being penetrated by a track with cross-sectional area δA_t traveling in direction $\mathbf{\Omega}$. The area of the penetrated surface area is $\delta A_t/(\mathbf{\Omega}\cdot\mathbf{n})$.

angular weights from Eq. 2.16, the integrated net current over the interfacial surface can be calculated using Eq. B.19.

$$\int\limits_{S\in S_{j,h}} dS\, J_g(\mathbf{r})\cdot\mathbf{n} = \sum_{(t,s)\in S_{j,h}} w_{i,t}\psi_g^{t,s}(s_{j,h}) \tag{B.19}$$

Similar to the CMFD cross-sections, these currents are formed from the MOC calculation, so they are only approximate until convergence. Summing over all MOC groups g within CMFD group e gives a representation for the net current $\tilde{J}_{j,h,e}$ across surface h of cell j for CMFD group e in Eq. B.20.

$$\tilde{J}_{j,h,e} = \sum_{g\in e}\sum_{(t,s)\in S_{j,h}} w_t\psi_g^{t,s}(s_{j,h}) \tag{B.20}$$

It is important to note that these estimates rely on angular fluxes. Since the entire angular flux vector is not stored explicitly, as discussed in Appendix A, when a CMFD surface is encountered during the MOC transport sweep, the contribution of angular fluxes along the track to the net current on the CMFD surface must be tallied. This is usually a relatively cheap operation, not adding much work to the transport sweep. With these calculated currents, the new transport equation is given in Eq. B.21.

$$\frac{1}{V_C^j}\sum_{h=1}^{H}\tilde{J}_{j,h,e} + \Sigma_{C,t}^{j,e}\phi_C^{j,e} = \frac{\chi_C^{j,e}}{k}\sum_{e'=1}^{E}\nu_C^{j,e'}\Sigma_{C,f}^{j,e'}\phi_C^{j,e'} + \sum_{e'=1}^{E}\Sigma_{C,s}^{i,e'\to e}\phi_C^{j,e'} \tag{B.21}$$

With this new representation of neutron balance, it is possible to solve for new scalar fluxes. However, this is not in the form of an eigenvalue problem since the streaming term has no dependence on the scalar flux. Therefore, new terms are introduced that relate the current to the scalar flux via diffusion coefficients. This is shown in Eq. B.22 [69].

$$\frac{\tilde{J}_{j,h,e}}{A_{j,h}} = -u(j,h)\hat{D}_{j,e}\left(\phi_C^{I(j,h),e} - \phi_C^{j,e}\right) - \tilde{D}_{j,h,e}\left(\phi_C^{I(j,h),e} + \phi_C^{j,e}\right) \tag{B.22}$$

The function $u(j,h)$ is the *sense* of the surface h on cell j, $A_{j,h}$ is the area of surface $S_{j,h}$, $\hat{D}_{j,e}$ is the surface diffusion coefficient, and $\tilde{D}_{j,h,e}$ is the nonlinear corrected diffusion coefficient. The inspiration of the first term involving $\hat{D}_{j,e}$ comes from diffusion theory. Specifically it can be calculated using Eq. B.23 under a CMFD uniform mesh assumption

$$\hat{D}_{j,h,e} = \frac{D_{j,e}D_{I(j,h),e}}{\Delta\mathbf{r}_h\left(D_{j,e} + D_{I(j,h),e}\right)} \tag{B.23}$$

where $\Delta\mathbf{r}_h$ is the distance between the CMFD cell and the interfacial surface h, the function $I(j,h)$ computes the index of the neighboring CMFD cell of j on surface h, and $D_{j,e}$ is the diffusion coefficient of cell j in group e. Due to the uniform mesh assumption, the distance between the centroid of cell j and surface h is the same as that of the neighboring cell $I(j,h)$. In this thesis, the uniform mesh assumption is imposed, but a more general treatment is possible. The diffusion coefficients are defined in Eq. B.24, with motivation from how diffusion coefficients are calculated in common nodal diffusion theory.

$$D_{j,e} = \frac{\sum_{i\in j}\sum_{g\in e}\frac{1}{3\Sigma_t^{i,g}}\overline{\phi_{i,g}}V_i}{\sum_{i\in j}\sum_{g\in e}\overline{\phi_{i,g}}V_i} \tag{B.24}$$

The sense $u(j,h)$ is calculated by Eq. B.25 where $\mathbb{1}$ is just the vector of ones in three dimensions and $\mathbf{n}_{j,h}$ is the normal vector of surface $S_{j,h}$. For a Cartesian uniform mesh, the sense is +1 if the surface is a positive x, y, or z surface and −1 if it is a negative x, y, or z surface.

$$u(j,h) = \frac{\mathbb{1} \cdot \mathbf{n}_{j,h}}{|\mathbb{1} \cdot \mathbf{n}_{j,h}|} \tag{B.25}$$

Lastly, the corrected diffusion coefficients $\tilde{D}_{j,h,e}$ are computed based on the relationship in Eq. B.22 for fluxes computed after the MOC transport sweep and before the CMFD solve. This makes the CMFD calculation consistent with the MOC calculation at convergence [69]. The calculation of the corrected diffusion coefficients therefore follows Eq. B.26 where $\tilde{\phi}_C^{j,e}$ are the CMFD cell-averaged scalar fluxes calculated from the MOC iteration with the tilde indicating the quantity comes from the MOC calculation and does not change throughout the CMFD iterations.

$$\tilde{D}_{j,h,e} = \frac{-u(j,h)\hat{D}_{j,h,e}\left(\tilde{\phi}_C^{I(j,h),e} - \tilde{\phi}_C^{j,e}\right) - \frac{\tilde{J}_{j,h,e}}{A_{j,h}}}{\tilde{\phi}_C^{I(j,h),e} + \tilde{\phi}_C^{j,e}} \tag{B.26}$$

Returning to balance equation and inserting the relationship for net current yields the new balance equation given in Eq. B.27.

$$\begin{aligned}\frac{1}{V_C^j}\sum_{h=1}^{H} A_{j,h}\left(-u(j,h)\hat{D}_{j,e}\left(\phi_C^{I(j,h),e} - \phi_C^{j,e}\right) - \tilde{D}_{j,h,e}\left(\phi_C^{I(j,h),e} + \phi_C^{j,e}\right)\right) + \Sigma_{C,t}^{j,e}\phi_C^{j,e} = \\ \frac{\chi_C^{j,e}}{k}\sum_{e'=1}^{E}\nu_C^{j,e'}\Sigma_{C,f}^{j,e'}\phi_C^{j,e'} + \sum_{e'=1}^{E}\Sigma_{C,s}^{i,e'\to e}\phi_C^{j,e'}\end{aligned} \tag{B.27}$$

B.3 Solving the CMFD Equations for MOC Acceleration

The CMFD balance equation in Eq. B.27 represents a physically equivalent system to solve the transport equation as the MOC balance equation in Eq. 2.24. Re-arranging terms in the balance equation and grouping by scalar flux yields the relationship in

Eq. B.28.

$$\left[\Sigma_{C,t}^{j,e}V_C^j + \sum_{h=1}^{H} A_{j,h}\left(u(j,h)\hat{D}_{j,e} - \tilde{D}_{j,h,e}\right)\right]\phi_C^{j,e} - \sum_{h=1}^{H} A_{j,h}\left(\tilde{D}_{j,h,e} + u(j,h)\hat{D}_{j,e}\right)\phi_C^{I(j,h),e}$$
$$- \sum_{e'=1}^{E} \Sigma_{C,s}^{i,e'\to e} V_C^j \phi_C^{j,e'} = \frac{\chi_C^{j,e}}{k} \sum_{e'=1}^{E} \nu_C^{j,e'} \Sigma_{C,f}^{j,e'} V_C^j \phi_C^{j,e'} \tag{B.28}$$

This can be written in terms of a matrix eigenvalue problem by defining a scalar flux vector Φ_C which incorporates all of the CMFD cell-averaged scalar fluxes $\phi_C^{j,e}$, a loss matrix A, and a multiplicative matrix M in Eq. B.29.

$$A\Phi_C = \frac{1}{k} M \Phi_C \tag{B.29}$$

This can be related to a regular eigenvalue problem by taking the inverse of A as

$$A^{-1} M \Phi_C = k \Phi_C. \tag{B.30}$$

Any common eigenvalue solver can be used to solve this system. In this thesis, simple power iteration is employed. This requires that every iteration solve a linear system. In this thesis, the linear system is solved with the red-black SOR algorithm [70]. The power iteration algorithm and its application to the CMFD equations is described later in Alg. B-1.

Often, a relaxation factor is applied to the corrected diffusion coefficients in order to ensure stability [71]. With the relaxation factor ω, the computed corrected diffusion coefficients are damped in iteration $n+1$ by

$$\tilde{D}_{j,h,e}^{n+1} = \omega \tilde{D}_{j,h,e}^{n+1/2} + (1-\omega)\tilde{D}_{j,h,e}^{n} \tag{B.31}$$

where the half-iterations refer to the computed diffusion coefficient without damping, as given in Eq. B.26. The relaxation factor ω can be any real number in the interval [0,1] chosen by the user. A lower relaxation factor leads to greater stability, but also leads to slower convergence.

B.4 Convergence Criteria

Convergence for source iteration is determined when `esp-MOC` defined in Eq. 7.17 is reduced to 10^{-4} and the change in eigenvalue estimate from the previous iteration is less than 1 pcm. The same criteria is imposed for convergence with CMFD accelerationn.

When CMFD acceleration is applied, there are also tolerances for the CMFD solver. Since the CMFD equations are solved with power iteration and during each iteration a linear system is solved with red-black SOR, two CMFD tolerances are required: the tolerance for power iteration and the tolerance for the linear solver.

If a constant tolerance were set, significant computational time could be wasted when the system is far from convergence. In the context of CMFD acceleration for MOC, early MOC source iterations will have significant error in neutron sources. Therefore, the resulting CMFD solution will also have significant error, even with strict convergence criteria. For this reason, the convergence criteria is always relative to either the current residual of the MOC solution or the reduction in error from the starting guess. In every iteration, the starting guess is the previously converged solution, and therefore also implicitly tied to the MOC residual.

In this thesis, the tolerance on CMFD power iteration is chosen to be an error reduction by a factor of 0.03 from the first iteration residual. For the linear solve, an error reduction by a factor of 0.1 is required *or* a neutron production residual of less than 0.01 `eps-MOC`. For both the power iteration and linear solve, a minimum of 25 iterations is enforced.

B.5 Prolongation

Once the CMFD equations are solved, the solution is used to update MOC fluxes, hence producing a new source for the next iteration. The updating of MOC fluxes is the prolongation step of the CMFD process. There are a variety of ways for which the CMFD fluxes can be updated. One simple approach is just updating all MOC fluxes in source region i and MOC group g encompassed by CMFD cell j and CMFD group e by applying Eq. B.32:

$$\phi_{i,g}^{\text{new}} = \phi_{i,g}^{\text{old}} \frac{\phi_{C,\text{new}}^{j,e}}{\phi_{C,\text{old}}^{j,e}} \tag{B.32}$$

where $\phi_{i,g}^{\text{new}}$ refers to the updated MOC flux after prolongation, $\phi_{i,g}^{\text{old}}$ refers to the MOC flux before prolongation, $\phi_{C,\text{old}}^{j,e'}$ refers to the CMFD cell-averaged flux at the start of the CMFD solution (calculated directly from the MOC fluxes), and $\phi_{C,\text{new}}^{j,e'}$ refers to the CMFD cell-averaged flux at convergence of the CMFD solution. It is important to ensure that both new and old scalar fluxes are normalized in the same manner.

While this prolongation approach works well when the CMFD mesh is fine, the CMFD acceleration could be aided by interpolating the CMFD solution if the shape is known. For the axial direction, we expect smoothly varying flux shapes due to the simplistic geometric axial structure. Therefore, the flux shape in each cell is approximated as being quadratic. Both the before and after CMFD flux shapes are fit with a quadratic interpolation using neighboring domains. The interpolant is chosen to preserve the average flux in each of the three nodes (the current cell and two axial neighbors). Specifically, the axial flux distribution a CMFD cell j for energy group e is represented as

$$\begin{aligned}\phi_C^{j,e}(z) \approx{}& \phi_C^{j-1,e}\left(\frac{\left(z - z_j^C\right)^2}{2} - \left(z - z_j^C\right) - \frac{1}{24}\right) + \phi_C^{j,e}\left(-\left(z - z_j^C\right)^2 + \frac{26}{24}\right) + \\ & \phi_C^{j+1,e}\left(\frac{\left(z - z_j^C\right)^2}{2} - \left(z - z_j^C\right) - \frac{1}{24}\right)\end{aligned} \tag{B.33}$$

for axial height z with the centroid of the CMFD cell at height z_j^C and CMFD cells $j-1$ and $j+1$ representing the lower and upper neighboring axial cells, respectively. For boundary CMFD cells on the domain, the expansion from the neighboring CMFD cell is used. The quadratic expansions are then used to update MOC fluxes as

$$\phi_{i,g}^{\text{new}} = \phi_{i,g}^{\text{old}} \frac{\phi_{C,\text{new}}^{j,e}(z_i^C)}{\phi_{C,\text{old}}^{j,e}(z_i^C)} \tag{B.34}$$

where z_i^C is the centroid of MOC source region i. If only two axial CMFD cells are present on the domain, then a linear fit is used instead of a quadratic fit. If only one axial CMFD cell is present on the domain, no fit is performed and MOC fluxes are updated with the simple relationship shown in Eq. B.32. A depiction of the axial fitting of CMFD fluxes is shown in Figure B-4.

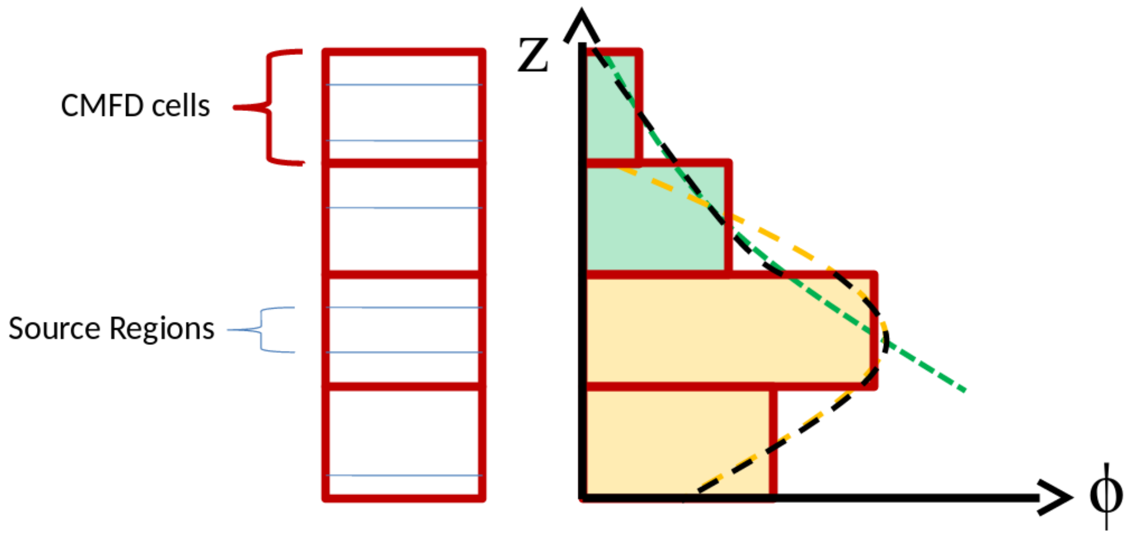


Figure B-4: A depiction of the axial prolongation for updating MOC fluxes with CMFD acceleration. The green dashed line shows the expansion in the top two axial CMFD cells and the orange dashed line shows the expansion used in the bottom two CMFD cells. The black dashed line shows the composite of the expansions.

Updating the MOC fluxes with the CMFD solution allows convergence to be greatly accelerated, quickly capturing the flux shape over the coarse mesh. In addition, the computational cost of fully converging the CMFD solution is small in comparison with just one MOC transport sweep. Therefore, using the process laid out at the beginning of this chapter in Figure B-1, the CMFD equations are formed and solved after every transport sweep. This process for solving the MOC neutron transport eigenvalue problem with CMFD acceleration is detailed in Alg. B-1.

Algorithm B-1: MOC Eigenvalue Solver with CMFD Acceleration

```
 1: procedure COMPUTEEIGENVALUE($geometry$, $tracks$, $N$)
 2:     Implicitly define $F$, $S$, $H$, $D$, and $T$ from $geometry$        ▷ Definitions in App. A
 3:     $\boldsymbol{\phi} \leftarrow \mathbb{1}$        ▷ Initialize scalar fluxes to all ones
 4:     $\boldsymbol{\psi} \leftarrow 0$        ▷ Initialize angular fluxes to zeros (only boundary stored)
 5:     $k \leftarrow 1$        ▷ Initialize the eigenvalue to 1
 6:     $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} / \left(\mathbb{1}^T F \boldsymbol{\phi}\right)$        ▷ Normalize by total fission source
 7:     while not converged do
 8:         $\mathbf{q} \leftarrow \frac{1}{4\pi}\left(S\boldsymbol{\phi} + \frac{1}{k}F\boldsymbol{\phi}\right)$        ▷ Compute neutron sources
 9:         $\boldsymbol{\psi} \leftarrow T^{-1}HD^{-1}\mathbf{q}$        ▷ Transport Sweep: these equations are
            $\mathbf{p} \leftarrow W\boldsymbol{\psi}$        solved simultaneously for computational efficiency. Currents on CMFD surfaces are tallied. Only angular fluxes on the boundary are explicitly stored.
10:         $\boldsymbol{\phi} \leftarrow D^{-1}\mathbf{q} + D^{-1}\mathbf{p}$        ▷ Scalar fluxes computed from tally **p**
11:         Form CMFD matrices $A$ and $M$        ▷ Defined by Eq. B.28 and Eq. B.29 (**restriction**)
12:         Compute CMFD scalar fluxes $\Phi_C$        ▷ Defined by Eq. B.15 (**restriction**)
13:         $\Phi_C \leftarrow \Phi_C / \left(\mathbb{1}^T M \Phi_C\right)$        ▷ Normalize by total CMFD fission rate
14:         $\tilde{\Phi}_C \leftarrow \Phi_C$        ▷ Save pre-CMFD scalar fluxes
15:         while not converged do
16:             $\Phi_C \leftarrow A^{-1} M \Phi_C$        ▷ Compute inverse with a linear solver (Gauss-Seidel)
17:             $k \leftarrow \mathbb{1}^T M \Phi_C$        ▷ Implicitly computing $\left(\mathbb{1}^T M A^{-1} M \Phi_C\right) / \left(\mathbb{1}^T M \Phi_C\right)$
18:             $\Phi_C \leftarrow \Phi_C / \left(\mathbb{1}^T M \Phi_C\right)$        ▷ Normalize by total CMFD fission rate
19:         end while
20:         Update $\boldsymbol{\phi}$ by interpolating the ratio of $\Phi_C$ and $\tilde{\Phi}_C$        ▷ This is the **prolongation** step
21:         $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} / \left(\mathbb{1}^T F \boldsymbol{\phi}\right)$        ▷ Normalize by total fission source
22:     end while
23:     return $\boldsymbol{\phi}$ and $k$        ▷ Return scalar fluxes and the eigenvalue
24: end procedure
```

Appendix C

Energy Group Structures

The energy group structures are from the CASMO-4 lattice physics code [63].

Table C.1: One group energy boundaries.

Group No.	Lower Bound [MeV]	Upper Bound [MeV]
1	0.0000E+00	2.0000E+01

Table C.2: Two group energy boundaries.

Group No.	Lower Bound [MeV]	Upper Bound [MeV]
2	0.0000E+00	6.2500E-07
1	6.2500E-07	2.0000E+01

Table C.3: Four group energy boundaries.

Group No.	Lower Bound [MeV]	Upper Bound [MeV]
4	0.0000E+00	6.2500E-07
3	6.2500E-07	5.5300E-03
2	5.5300E-03	8.2100E-01
1	8.2100E-01	2.0000E+01

Table C.4: Eight group energy boundaries.

Group No.	Lower Bound [MeV]	Upper Bound [MeV]
8	0.0000E+00	5.8000E-08
7	5.8000E-08	1.4000E-07
6	1.4000E-07	2.8000E-07
5	2.8000E-07	6.2500E-07
4	6.2500E-07	4.0000E-06
3	4.0000E-06	5.5300E-03
2	5.5300E-03	8.2100E-01
1	8.2100E-01	2.0000E+01

Table C.5: Eleven group energy boundaries.

Group No.	Lower Bound [MeV]	Upper Bound [MeV]
11	0.0000E+00	5.8000E-08
10	5.8000E-08	1.4000E-07
9	1.4000E-07	2.8000E-07
8	2.8000E-07	6.2500E-07
7	6.2500E-07	4.0000E-06
6	4.0000E-06	9.8770E-06
5	9.8770E-06	1.5968E-05
4	1.5968E-05	2.7700E-05
3	2.7700E-05	5.5300E-03
2	5.5300E-03	8.2100E-01
1	8.2100E-01	2.0000E+01

Table C.6: Sixteen group energy boundaries.

Group No.	Lower Bound [MeV]	Upper Bound [MeV]
16	0.0000E+00	3.0000E-08
15	3.0000E-08	5.8000E-08
14	5.8000E-08	1.4000E-07
13	1.4000E-07	2.8000E-07
12	2.8000E-07	3.5000E-07
11	3.5000E-07	6.2500E-07
10	6.2500E-07	8.5000E-07
9	8.5000E-07	9.7200E-07
8	9.7200E-07	1.0200E-06
7	1.0200E-06	1.0970E-06
6	1.0970E-06	1.1500E-06
5	1.1500E-06	1.3000E-06
4	1.3000E-06	4.0000E-06
3	4.0000E-06	5.5300E-03
2	5.5300E-03	8.2100E-01
1	8.2100E-01	2.0000E+01

Table C.7: Twenty-five group energy boundaries.

Group No.	Lower Bound [MeV]	Upper Bound [MeV]
25	0.0000E+00	3.0000E-08
24	3.0000E-08	5.8000E-08
23	5.8000E-08	1.4000E-07
22	1.4000E-07	2.8000E-07
21	2.8000E-07	3.5000E-07
20	3.5000E-07	6.2500E-07
19	6.2500E-07	9.7200E-07
18	9.7200E-07	1.0200E-06
17	1.0200E-06	1.0970E-06
16	1.0970E-06	1.1500E-06
15	1.1500E-06	1.8550E-06
14	1.8550E-06	4.0000E-06
13	4.0000E-06	9.8770E-06
12	9.8770E-06	1.5968E-05
11	1.5968E-05	1.4873E-04
10	1.4873E-04	5.5300E-03
9	5.5300E-03	9.1180E-03
8	9.1180E-03	1.1100E-01
7	1.1100E-01	5.0000E-01
6	5.0000E-01	8.2100E-01
5	8.2100E-01	1.3530E+00
4	1.3530E+00	2.2310E+00
3	2.2310E+00	3.6790E+00
2	3.6790E+00	6.0655E+00
1	6.0655E+00	2.0000E+01

Table C.8: Seventy group energy boundaries.

Group No.	Lower Bound [MeV]	Upper Bound [MeV]
70	0.0000E+00	5.0000E-09
69	5.0000E-09	1.0000E-08
68	1.0000E-08	1.5000E-08
67	1.5000E-08	2.0000E-08
66	2.0000E-08	2.5000E-08
65	2.5000E-08	3.0000E-08
64	3.0000E-08	3.5000E-08
63	3.5000E-08	4.2000E-08
62	4.2000E-08	5.0000E-08
61	5.0000E-08	5.8000E-08
60	5.8000E-08	6.7000E-08
59	6.7000E-08	8.0000E-08
58	8.0000E-08	1.0000E-07
57	1.0000E-07	1.4000E-07
56	1.4000E-07	1.8000E-07
55	1.8000E-07	2.2000E-07
54	2.2000E-07	2.5000E-07
53	2.5000E-07	2.8000E-07
52	2.8000E-07	3.0000E-07
51	3.0000E-07	3.2000E-07
50	3.2000E-07	3.5000E-07
49	3.5000E-07	4.0000E-07
48	4.0000E-07	5.0000E-07
47	5.0000E-07	6.2500E-07
46	6.2500E-07	7.8000E-07
45	7.8000E-07	8.5000E-07
44	8.5000E-07	9.1000E-07
43	9.1000E-07	9.5000E-07
42	9.5000E-07	9.7200E-07
41	9.7200E-07	9.9600E-07
40	9.9600E-07	1.0200E-06
39	1.0200E-06	1.0450E-06
38	1.0450E-06	1.0710E-06
37	1.0710E-06	1.0970E-06
36	1.0970E-06	1.1230E-06
35	1.1230E-06	1.1500E-06
34	1.1500E-06	1.3000E-06
33	1.3000E-06	1.5000E-06
32	1.5000E-06	1.8550E-06
31	1.8550E-06	2.1000E-06
30	2.1000E-06	2.6000E-06
29	2.6000E-06	3.3000E-06
28	3.3000E-06	4.0000E-06
27	4.0000E-06	9.8770E-06
26	9.8770E-06	1.5968E-05

25	1.5968E-05	2.7700E-05
24	2.7700E-05	4.8052E-05
23	4.8052E-05	7.5501E-05
22	7.5501E-05	1.4873E-04
21	1.4873E-04	3.6726E-04
20	3.6726E-04	9.0690E-04
19	9.0690E-04	1.4251E-03
18	1.4251E-03	2.2395E-03
17	2.2395E-03	3.5191E-03
16	3.5191E-03	5.5300E-03
15	5.5300E-03	9.1180E-03
14	9.1180E-03	1.5030E-02
13	1.5030E-02	2.4780E-02
12	2.4780E-02	4.0850E-02
11	4.0850E-02	6.7340E-02
10	6.7340E-02	1.1100E-01
9	1.1100E-01	1.8300E-01
8	1.8300E-01	3.0250E-01
7	3.0250E-01	5.0000E-01
6	5.0000E-01	8.2100E-01
5	8.2100E-01	1.3530E+00
4	1.3530E+00	2.2310E+00
3	2.2310E+00	3.6790E+00
2	3.6790E+00	6.0655E+00
1	6.0655E+00	2.0000E+01

Appendix D

On-the-fly Ray Tracing by z-Stack

In Chapter 5, on-the-fly ray tracing was introduced. In this appendix, the relationships formed for calculating segment crossings of a z-stack are more thoroughly explained.

Recall that all tracks in a z-stack have the same polar angle θ, project onto the same 2D track, and are separated by a constant axial ray spacing δz. Therefore, the axial height z_i of the i^{th} lowest track (starting from 0) can be given as a function of distance s along the associated 2D track as

$$z_i(s) = z_0(0) + i\delta z + s \cot\theta \tag{D.1}$$

where $z_0(0)$ is the z-coordinate at the intersection of the lowest track with the z-axis at the start of the associated 2D track.

For each 2D segment in the 2D track, there is an associated axially extruded region which contains a list of 3D SRs in the region. Using the boundaries of each SR and Eq. D.1, it is possible to analytically compute the indexes in the z-stack of tracks that will cross the SR.

To begin, the first track to traverse the SR (lowest index i) is determined. All tracks that traverse the SR must have their highest z-height above the lowest boundary of the SR, z_{min}. Specifically,

$$\max_{s_{start} \le s \le s_{end}} z_i(s) > z_{min} \tag{D.2}$$

where s_{start} is the s distance along the 2D track at the start of the 2D segment and s_{end} is the s distance along the 2D track at the end of the 2D segment. Inserting the relationship in Eq. D.1, this can be expanded to

$$\max_{s_{start} \leq s \leq s_{end}} z_0(0) + i\delta z + s\cot\theta > z_{min}. \tag{D.3}$$

Noting the linear relationship of the tracks and the constant axial ray spacing, this can be simplified in terms of the lowest track with height z_0 as

$$\max\left(z_0(s_{start}), z_0(s_{end})\right) + i\delta z > z_{min} \tag{D.4}$$

and cast in terms of the index i as

$$i > \frac{z_{min} - \max\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z}. \tag{D.5}$$

Since i is an integer, the lowest track index i_{start} to traverse the region can be given by

$$i_{start} = \left\lceil \frac{z_{min} - \max\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z} \right\rceil \tag{D.6}$$

Next, the last track to traverse the SR (highest index i) is determined. All tracks that traverse the SR must have their lowest z-height below the highest boundary of the SR, z_{max}. Specifically,

$$\min_{s_{start} \leq s \leq s_{end}} z_i(s) < z_{max} \tag{D.7}$$

which can be expanded to

$$\min_{s_{start} \leq s \leq s_{end}} z_0(0) + i\delta z + s\cot\theta < z_{max}. \tag{D.8}$$

Similar to the arguments for the first track index, the last track index follows the criteria

$$i < \frac{z_{max} - \min\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z}. \tag{D.9}$$

Therefore, the last track to cross the SR has index i_{end} given by

$$i_{end} = \left\lfloor \frac{z_{max} - \min\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z} \right\rfloor. \tag{D.10}$$

In addition to analytically calculating the first and last track indexes, it is possible to calculate the indexes of tracks with the same length. Specifically, it is possible to calculate the first and last tracks to cross the entire 2D segment length or the entire 3D source height.

For a track to cross the entire 2D segment length, both its beginning and ending heights must be between the minimum and maximum SR boundaries. For a track to cross the entire axial height of the SR, its beginning and ending heights must be above and below the minimum and maximum SR boundaries, respectively.

Two indexes i_{in} and i_{out} are calculated. The index i_{in} represents the first track to start above the lowest SR boundary and i_{out} represents the first track to end above the maximum SR boundary. All tracks with index between i_{in} and i_{out} traverse the entire 2D segment length. All tracks with index between i_{out} and i_{in} will traverse the entire SR height. Notice that these are mutually exclusive: if some tracks traverse the entire 2D segment length, none will traverse the entire SR height.

Tracks that start above the minimum boundary satisfy

$$\min_{s_{start} \leq s \leq s_{end}} z_i(s) > z_{min} \tag{D.11}$$

and tracks that end above the maximum boundary satisfy

$$\max_{s_{start} \leq s \leq s_{end}} z_i(s) > z_{max}. \tag{D.12}$$

Therefore, in a similar process to determining the first and last tracks to cross the SR, the indexes i_{in} and i_{out} can be calculated as:

$$i_{in} = \left\lceil \frac{z_{min} - \min\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z} \right\rceil \tag{D.13}$$

$$i_{out} = \left\lceil \frac{z_{max} - \max\left(z_0(s_{start}), z_0(s_{end})\right)}{\delta z} \right\rceil \tag{D.14}$$

Appendix E

The BEAVRS Benchmark

The results presented in this thesis are based on models formed from the BEAVRS benchmark [72]. This appendix introduces the BEAVRS benchmark in Section E.1, including a description of axial alterations made to the benchmark. Section E.2 details the particular models formed from cutouts of the BEAVRS model.

E.1 Introduction to the BEAVRS Benchmark

The BEAVRS benchmark [72] was released in 2013, representing a Westinghouse 4-loop nuclear power reactor. This reactor is representative of common PWR designs in the United States. The benchmark contains core loadings and detector measurements for the first two cycles of operation, but this thesis concentrates on Hot Zero Power (HZP) simulations at the beginning of the first cycle in an all rods out configuration.

The reactor contains 193 fuel assemblies. Each assembly contains a 17×17 lattice of fuel rods, guide tubes, and instrument tubes. The pin-pitch is 1.26 cm inside each assembly. All fuel rods within the same assembly contain uranium of the same enrichment. In the first cycle, three uranium enrichments are used: 1.6%, 2.4%, and 3.1%. Wet annular burnable absorbers are present throughout the core to flatten the power distribution. The active fuel height is 365.76 cm. A radial description of the core is shown in Figure E-1.

One of the goals of this thesis is to simulate the BEAVRS benchmark using the explicit

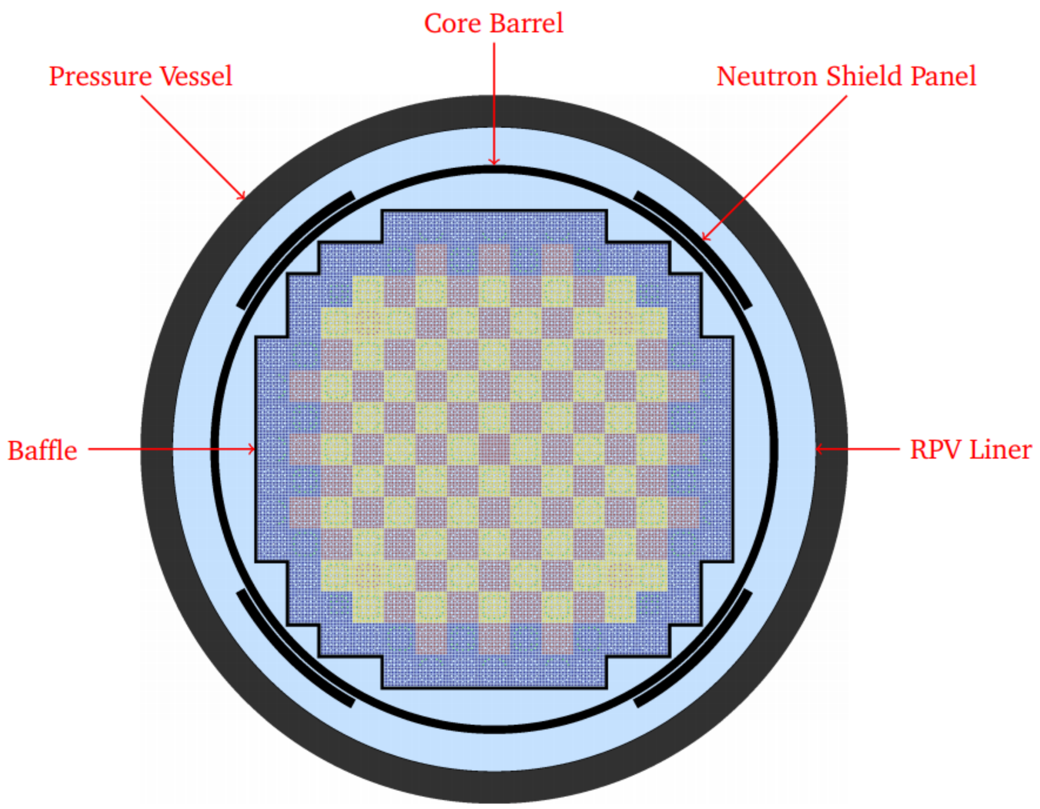
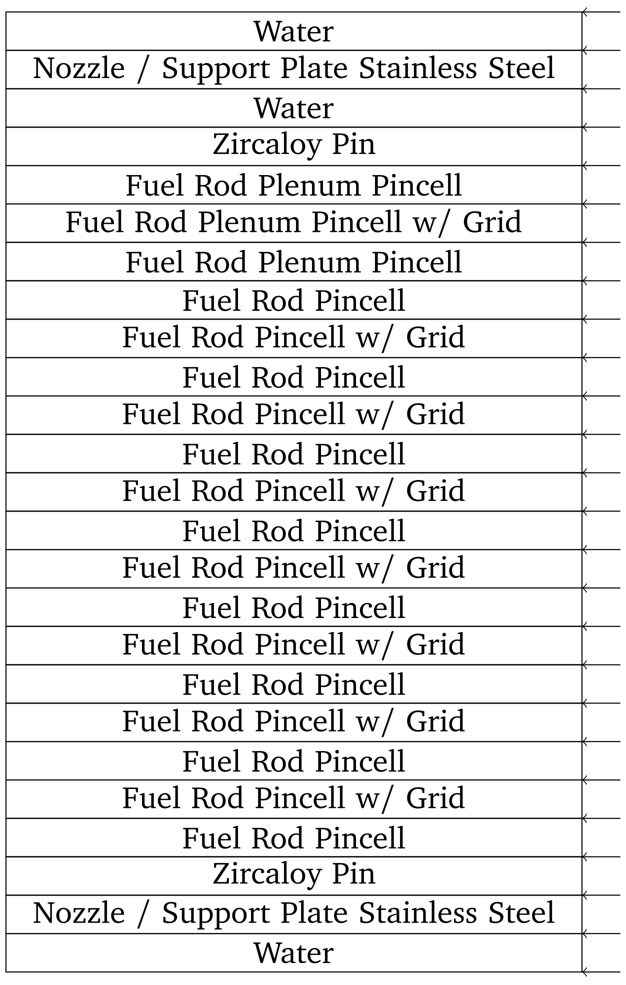


Figure E-1: A radial illustration of the BEAVRS benchmark with fuel pins colored by enrichment.

detail provided in the BEAVRS specification. However, in order to conduct uniform axial mesh refinement studies, the axial heights of material regions are altered such that each material discontinuity occurs at an even number of cm. The top and bottom grid spacers in the BEAVRS benchmark are 3.36 cm with intermediate grid spacers 5.72 cm. These lengths were altered to 2.0 cm and 6.0 cm, respectively. The starting height of the grid spacers were rounded to the nearest even integer. All other z-heights in the geometry were similarly rounded to the nearest even integer. The altering of axial heights allows regions to be formed which all have the same axial height, which simplifies axial uniform mesh refinement sensitivity studies. The altered axial heights are depicted in Figure E-2. While these alterations do change the benchmark slightly, and therefore also change the computed solutions, these solutions are still very close to those of the true BEAVRS benchmark.

Although the BEAVRS model is defined inside a cylindrical geometry, a rectangular



Elevation [BEAVRS] (cm)	Description
460 [460.000]	Highest Extent
432 [431.876]	Top of Upper Nozzle
424 [423.049]	Bottom of Upper Nozzle
420 [419.704]	Top of Fuel Rod
418 [417.164]	Top of Fuel Rod Plenum
414 [415.164]	Grid 8 Top
412 [411.806]	Grid 8 Bottom
402 [402.508]	Top of Active Fuel
366 [364.725]	Grid 7 Top
360 [359.010]	Grid 7 Bottom
312 [312.528]	Grid 6 Top
306 [306.813]	Grid 6 Bottom
260 [260.331]	Grid 5 Top
254 [254.616]	Grid 5 Bottom
208 [208.134]	Grid 4 Top
202 [202.419]	Grid 4 Bottom
156 [155.937]	Grid 3 Top
150 [150.222]	Grid 3 Bottom
104 [103.740]	Grid 2 Top
98 [98.025]	Grid 2 Bottom
40 [40.520]	Grid 1 Top
38 [37.162]	Grid 1 Bottom
36 [36.748]	Bottom of Active Fuel
34 [35.000]	Bottom of Fuel Rod
20 [20.000]	Bottom of Support Plate
0 [0.000]	Lowest Extent

Figure E-2: Fuel rod pincell axial specification.

bounding geometry is often used in MOC methods for cyclic tracking. Therefore, the BEAVRS model is modeled with a rectangular prism bounding the geometry. In the radial plane, the bounding dimensions are square with sides equal to 17 assembly widths. Since the BEAVRS model has a maximum of 15 assemblies along each x and y direction, this allows at least one assembly of radial reflector to be modeled outside the core. In addition, the corners have very deep water reflectors. In the axial direction, the BEAVRS benchmark is modeled over a height of 400 cm, from 20 cm to 420 cm in the model specification, allowing approximately 20 cm of axial reflector in each direction. Vacuum boundaries are assumed on all surfaces.

Cross-sections are generated for each unique material in the BEAVRS model. The cross-section generation procedures are discussed in Appendix F. A plot of the BEAVRS benchmark colored by unique material region is shown in Figure E-3. In addition, an axial plot of the materials is shown in Figure E-4. Similarly the radial and axial material plots of a 1.6% enriched fuel assembly are shown in Figure E-5 and Figure E-6, respectively.

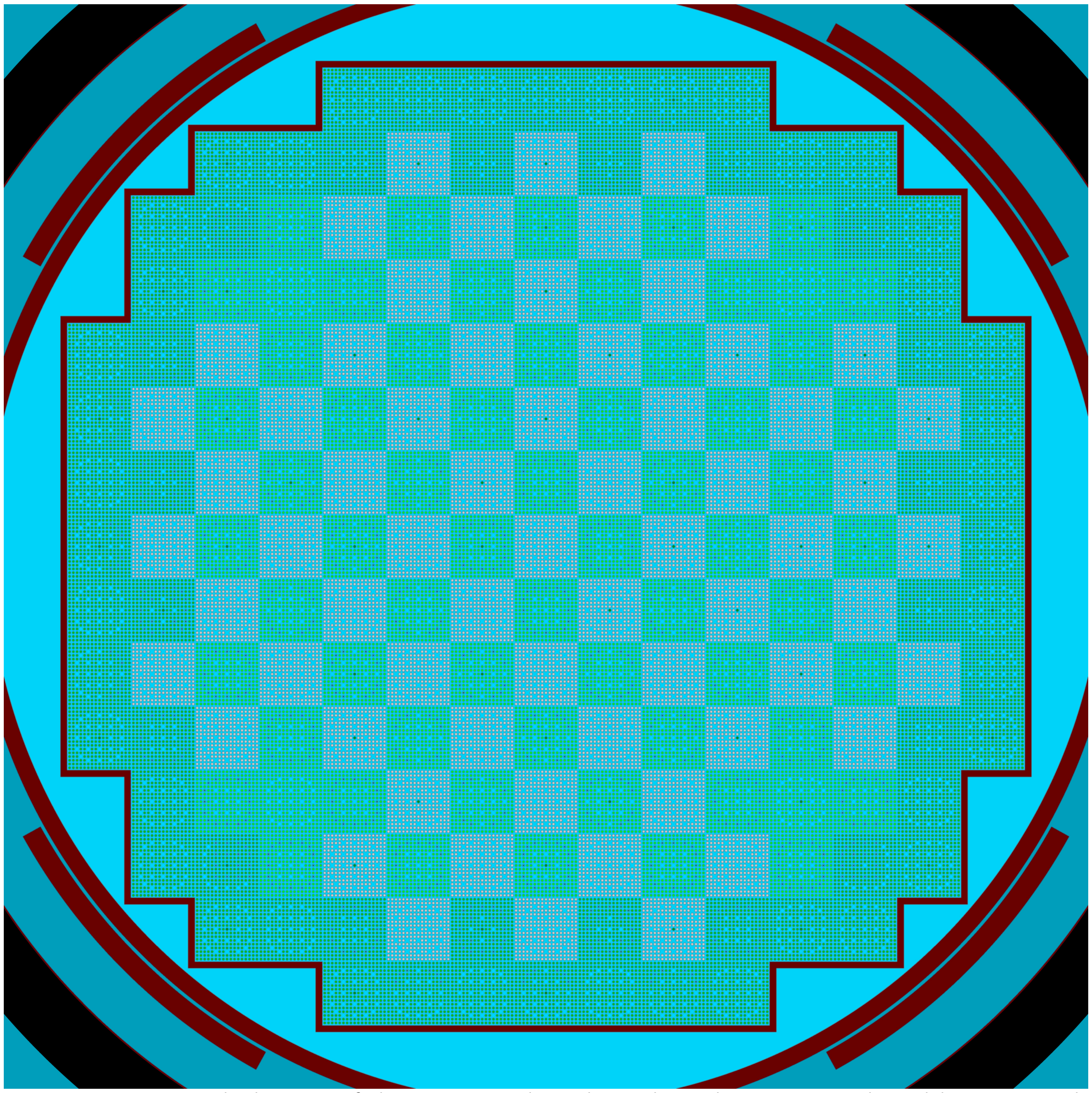


Figure E-3: A radial view of the BEAVRS benchmark with regions colored by material.

E.2 Description of BEAVRS Models

All models which are simulated in this thesis are derived from cutouts of the BEAVRS benchmark with 70 group cross-sections, as described in Appendix F. Cutouts are formed in order to evaluate smaller problems which are representative of computational performance or physical behavior of the large full core problem.

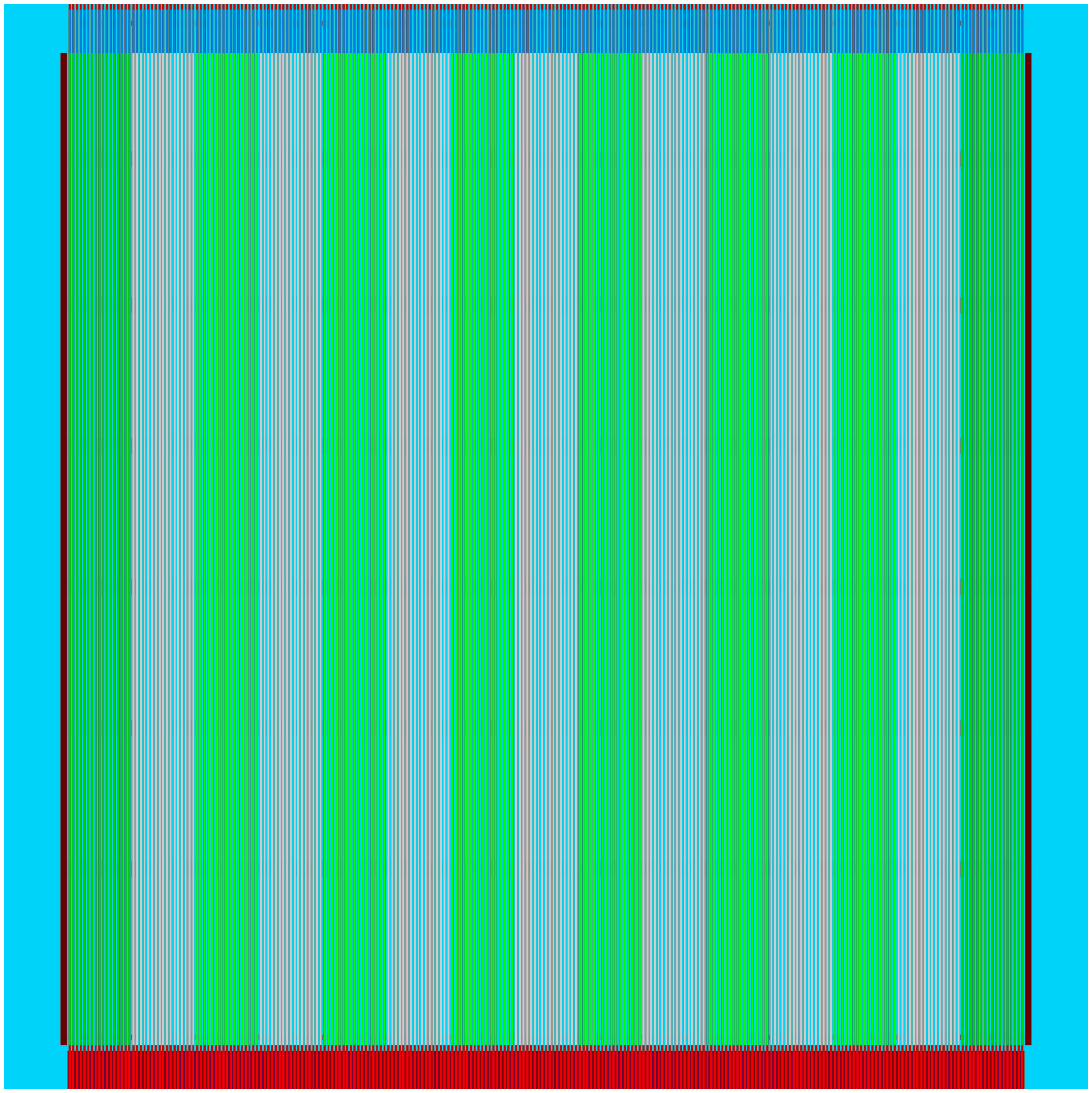


Figure E-4: An axial view of the BEAVRS benchmark with regions colored by material.

E.2.1 Full Core 3D Model

The first model is the full core 3D BEAVRS model, incorporating all the details discussed in previous sections. The simulation of this model directly corresponds with the OpenMC simulations from which the cross-sections are derived.

E.2.2 Full Core 2D Model

A 2D extruded cutout of the reactor is formed in order to determine the physical behavior of the BEAVRS model in the radial direction. This cutout is taken from a 10 cm axial

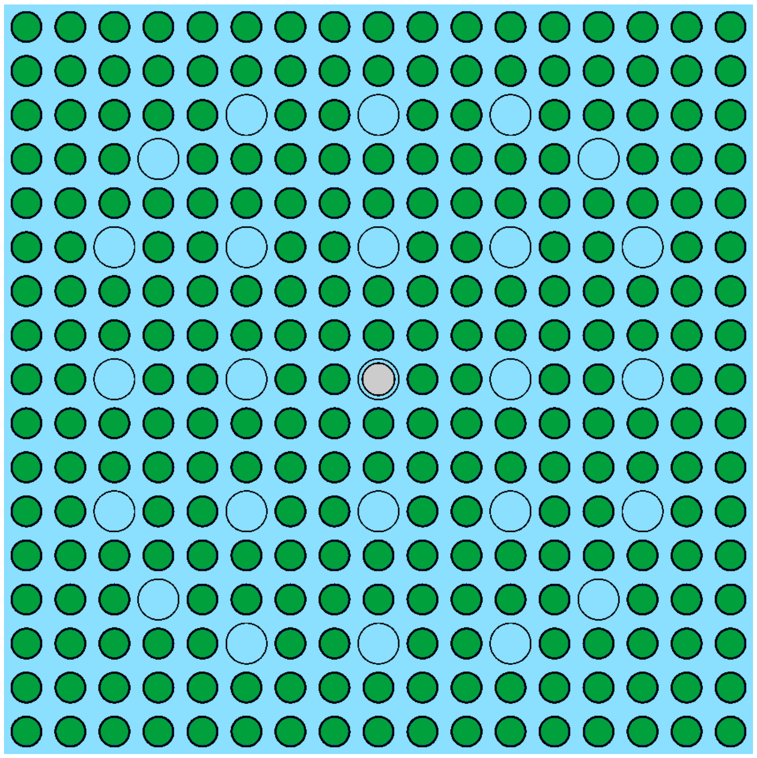


Figure E-5: A radial view of the 1.6% enriched fuel assembly in the BEAVRS benchmark with regions colored by material.

interval over which there are no grid spacers. Reflective boundary conditions are placed on the top and bottom of the problem. While this model lacks any axial variation, it still contains all radial detail except grid spacers. Due to the lack of axial detail, 2D MOC and 3D MOC simulations with sufficient parameter refinement should produce equivalent solutions. Only this model and the full core 3D model contain the full radial water reflector. Therefore, this model is very useful in determining the effect of large radial water reflectors.

E.2.3 Single Assembly Model

A single assembly model is formed which represents the full axial detail of a single 1.6% enriched fuel assembly. While this model lacks radial water reflectors, it contains the full axial detail of the full core problem, including grid spacers. Outside the core, full geometrical detail is also captured including support plate / nozzles and, most notably, water reflectors of approximately 20 cm above and below the fuel. Reflective boundaries

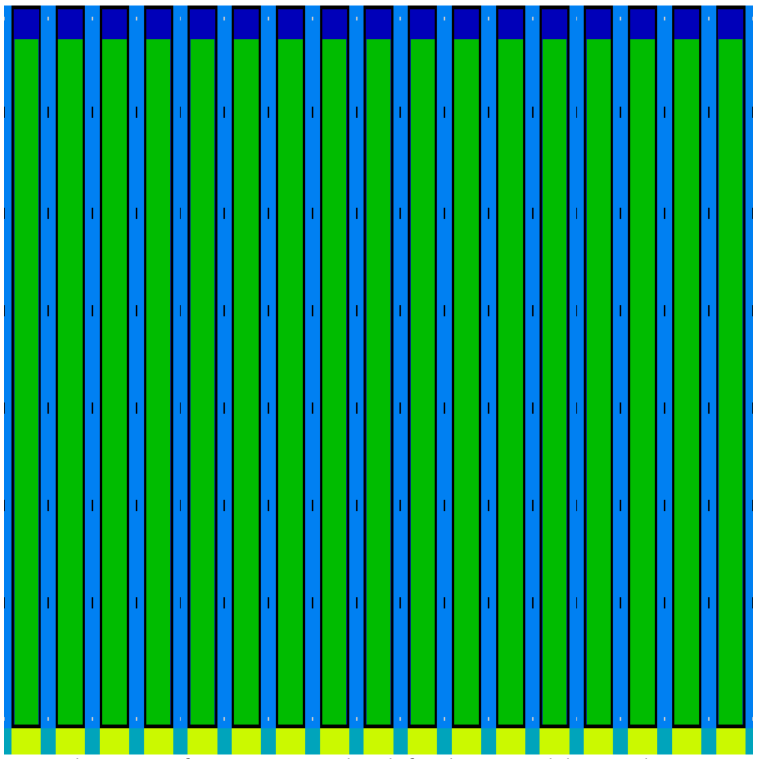


Figure E-6: An axial view of 1.6% enriched fuel assembly in the BEAVRS benchmark with regions colored by material.

are placed on the x and y boundaries. Physically, this is equivalent to an infinite 2D lattice of 1.6% enriched fuel assemblies. Vacuum boundaries remain on the top and bottom of this model.

E.2.4 Single Assembly Model without Reflectors

In addition to the single assembly model, a single assembly model without reflectors is also formed which contains all the detail of the single assembly model, but without the axial water reflectors. Specifically, 20 cm are removed from both the bottom and top of the model, resulting in a model that only covers the active fuel and is 360 cm tall. Axial boundaries conditions are vacuum.

E.2.5 SDSA Model

The Single Domain Single Assembly (SDSA) model represents a 20 cm tall cutout within the single assembly model which contains no grid spacers. Reflective boundary conditions are imposed on all surfaces.

E.2.6 Short Single Assembly Model

Similar to the SDSA model, the short single assembly model is created which allows for more feasible testing due to its far reduced size. This model is the same as the SDSA model except it is only 10 cm in axial height and contains 3.1% enriched fuel. This enrichment is the highest enrichment found in the cycle 1 BEAVRS model. The greater fuel enrichment allows for slightly larger gradients with a flux peak in the moderator.

E.2.7 Rodded Single Assembly Model

The rodded single assembly model is the only model which uses a geometry not explicitly found in the full core 3D BEAVRS model. The model is constructed in the same way as the single assembly model described in Section E.2.3, but with 3.1% enriched fuel and with all rods inserted, covering approximately half of the active fuel height. The axial zones of guide tubes containing the inserted control rods are shown in Figure E-7.

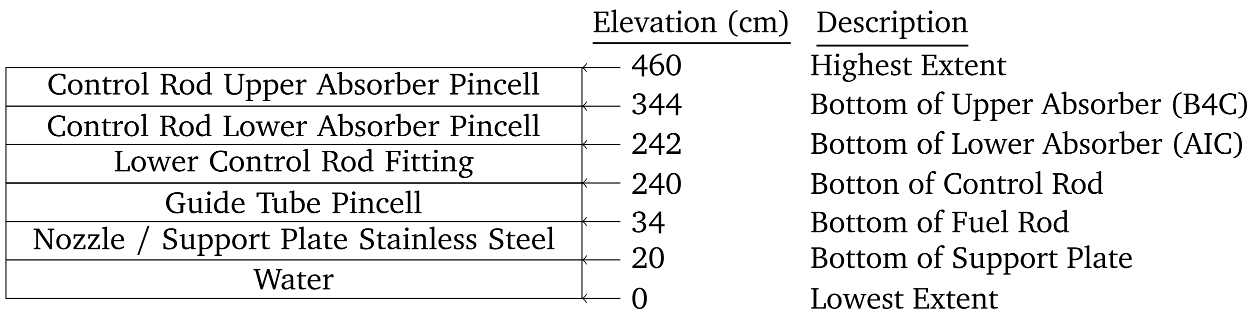

Elevation (cm)
Description
Control Rod Upper Absorber Pincell
Control Rod Lower Absorber Pincell
Lower Control Rod Fitting
Guide Tube Pincell
Nozzle / Support Plate Stainless Steel
Water
460
Highest Extent
344
Bottom of Upper Absorber (B4C)
242
Bottom of Lower Absorber (AIC)
240
Botton of Control Rod
34
Bottom of Fuel Rod
20
Bottom of Support Plate
0
Lowest Extent

Figure E-7: Control rod pincell axial specification for the single assembly control rod insertion model.

The large control rod insertion causes significant gradients within the axial scalar flux distribution, allowing for robust testing of 3D MOC on problems with significant axial variation. As mentioned previously, this model uses a separate cross-section library. Instead of simulating the all rods out configuration, which lacks control rods within the core, the rodded single assembly model is explicitly simulated in OpenMC to form cross-section estimates. This allows reasonable estimates of control rod material cross-sections.

Appendix F

Cross-section Generation

In this thesis, the same 70 group cross-section library is used for all results involving the BEAVRS benchmark or cutouts of the BEAVRS benchmark, except for rod insertion studies in which a separate 70 group cross-section library was formed in order to have accurate control rod material cross-sections. All cross-section libraries and group structures use the CASMO-4 energy group boundaries [63], as given in Appendix C. In this appendix, the process used to form cross-sections is thoroughly discussed.

This appendix is split into four sections. First, the basics of multi-group cross-section generation are discussed in Section F.1. Then the subtleties of angular dependence of total cross-sections and the formation of transport-corrected cross-sections are discussed in Section F.2 and Section F.3, respectively. Finally, the process to generate cross-sections using OpenMC is discussed in Section F.4

F.1 Cross-section Generation

The multi-group transport equation yields solutions equivalent to those from the continuous energy transport equation if cross-sections are appropriately defined. This is obtained for a given energy group g by integrating the continuous energy transport equation over the associated energy range of $[E_g, E_{g-1}]$. The resulting multi-group cross-section definitions are:

$$\Sigma_t^g(\mathbf{r},\boldsymbol{\Omega}) = \frac{\int_{E_g}^{E_{g-1}} dE\, \Sigma_t(\mathbf{r},E)\psi(\mathbf{r},\boldsymbol{\Omega},E)}{\int_{E_g}^{E_{g-1}} dE\, \psi(\mathbf{r},\boldsymbol{\Omega},E)} \tag{F.1}$$

$$\nu\Sigma_f^g(\mathbf{r}) = \frac{\int_{E_{g'}}^{E_{g'-1}} dE\ \nu(\mathbf{r},E)\Sigma_f(\mathbf{r},E)\phi(\mathbf{r},E)}{\int_{E_{g'}}^{E_{g'-1}} dE\, \phi(\mathbf{r},E)} \tag{F.2}$$

$$\chi_g(\mathbf{r}) = \frac{\int_{E_{g'}}^{E_{g'-1}} dE\, \chi(\mathbf{r},E) \sum_{g'=1}^{G} \nu\Sigma_f^{g'}(\mathbf{r})\phi_{g'}(\mathbf{r})}{\sum_{g=1}^{G} \nu\Sigma_f^g(\mathbf{r})\phi_g(\mathbf{r})} \tag{F.3}$$

$$\Sigma_s^{g'\to g}(\mathbf{r}) = \frac{\int_{E_g}^{E_{g-1}} dE\, \int_{E_{g'}}^{E_{g'-1}} dE'\, \Sigma_s(\mathbf{r},E'\to E)\phi(\mathbf{r},E')}{\int_{E_g}^{E_{g-1}} dE\, \phi(\mathbf{r},E)} \tag{F.4}$$

While seemingly straightforward, these equations require knowledge of the neutron fluxes, which are the goal of transport simulations. Therefore, approximations need to be made for the fluxes. One way to implicitly do this is through Monte Carlo simulations in which reaction rates are tallied. These tallied reaction rates can be used in conjunction with a tallied scalar flux estimate to produce multi-group cross-sections.

F.2 Angular Dependence of Total Cross-Sections

From the definition of the multi-group total cross section Σ_t^g in Eq. F.1 as

$$\Sigma_t^g(\mathbf{r},\boldsymbol{\Omega}) = \frac{\int_{E_g}^{E_{g-1}} dE\, \Sigma_t(\mathbf{r},E)\psi(\mathbf{r},\boldsymbol{\Omega},E)}{\int_{E_g}^{E_{g-1}} dE\, \psi(\mathbf{r},\boldsymbol{\Omega},E)},$$

it is clear that the multi-group total cross-section should be angular dependent even though we assume the continuous energy total cross-section is angular independent. This is because the total cross-section multiplies the angular flux rather than the scalar flux. In turn, this causes its collapsed multi-group form to be angular dependent since both the numerator and denominator of the expression in Eq. F.1 are angular dependent. This would be true of any cross-section that multiplies the angular flux, but with the isotropic scattering approximation, there are no other terms that multiply cross-sections by the angular flux.

This angular dependence is often neglected, but can introduce a bias [65]. With the angular dependence ignored, the relationship can be integrated over all directions leading to the form in Eq. F.5.

$$\Sigma_t^g(\mathbf{r}) = \frac{\int_{E_g}^{E_{g-1}} dE\, \Sigma_t(\mathbf{r}, E)\phi(\mathbf{r}, E)}{\int_{E_g}^{E_{g-1}} dE\, \phi(\mathbf{r}, E)} \tag{F.5}$$

The work in this thesis also relies on the approximation of angular independent multi-group total cross-sections. With this approximation, the new multi-group transport equation can be formed, as given in Eq. F.6 whose solution is the subject of this thesis.

$$\mathbf{\Omega}\cdot\nabla\psi_g(\mathbf{r},\mathbf{\Omega}) + \Sigma_t^g(\mathbf{r})\psi_g(\mathbf{r},\mathbf{\Omega}) = \frac{1}{4\pi}\left(\frac{\chi_g(\mathbf{r})}{k}\sum_{g'=1}^{G}\nu_{g'}(\mathbf{r})\,\Sigma_f^{g'}(\mathbf{r})\,\phi_{g'}(\mathbf{r}) + \sum_{g'=1}^{G}\Sigma_s^{g'\to g}(\mathbf{r})\,\phi_{g'}(\mathbf{r})\right) \tag{F.6}$$

It is important to remember that some error is expected from the absence of angular dependent total cross-sections so that the multi-group transport solution does not strictly match the corresponding continuous energy transport solution, such as that computed by Monte Carlo methods.

F.3 The Transport Correction

Previously it was mentioned that the assumption of isotropic scattering introduces significant bias, but is remedied by a transport correction. In this section, the transport correction is derived and its implications are discussed. Many different transport corrections have been implemented, such as those described in the TRANSX [73] and NJOY [74] manuals. The basis for these transport corrections was formulated by Bell, Hansen and Sandmeier [36]. However, this section largely follows Hebert's derivation [37].

The scattering term, without the isotropic assumption, follows the relationship

$$\int_0^\infty dE' \int_{4\pi} d\mathbf{\Omega}' \Sigma_s(\mathbf{r}, \mathbf{\Omega}' \to \mathbf{\Omega}, E' \to E)\psi(\mathbf{r}, \mathbf{\Omega}', E'). \tag{F.7}$$

In the laboratory system, in which neutron behavior is modeled, scattering may be strongly anisotropic. However, in the center-of-momentum framework, scattering is nearly isotropic in the energy range of interest. This implies that the angular dependence relies on the magnitude of the deflection from scattering $\mathbf{\Omega} \cdot \mathbf{\Omega}'$, reducing this relationship to

$$\int_0^\infty dE' \int_{4\pi} d\mathbf{\Omega}' \Sigma_s(\mathbf{r}, \mathbf{\Omega} \cdot \mathbf{\Omega}', E' \to E)\psi(\mathbf{r}, \mathbf{\Omega}', E').$$

The scattering kernel can be expressed as an expansion of Legendre polynomials P_ℓ with angular-independent coefficients $\Sigma_{s,\ell}(\mathbf{r}, E' \to E)$ as shown in Eq. F.8 where $\mu = \mathbf{\Omega} \cdot \mathbf{\Omega}'$.

$$\Sigma_s(\mathbf{r}, \mu, E' \to E) = \sum_{\ell=0}^{\infty} \frac{2\ell+1}{2} \Sigma_{s,\ell}\left(\mathbf{r}, E' \to E\right) P_\ell(\mu) \tag{F.8}$$

The coefficients can be determined by taking advantage of the orthogonality of Legendre polynomials, leading to the relationship in Eq. F.9.

$$\Sigma_{s,\ell}\left(\mathbf{r}, E' \to E\right) = \int_{-1}^{1} d\mu \, \Sigma_s(\mathbf{r}, \mu, E' \to E) P_\ell(\mu) \tag{F.9}$$

The scattering kernel can be approximated by a finite number L of Legendre polynomials with modified coefficients $\tilde{\Sigma}_{s,\ell}(\mathbf{r}, E' \to E)$ and a transport correction term $\Delta\Sigma_{tr}(\mathbf{r}, E' \to E)$ in Eq. F.10.

$$\Sigma_s(\mathbf{r}, \mu, E' \to E) \approx \sum_{\ell=0}^{L} \frac{2\ell+1}{2} \tilde{\Sigma}_{s,\ell}\left(\mathbf{r}, E' \to E\right) P_\ell(\mu) + \Delta\Sigma_{tr}\left(\mathbf{r}, E' \to E\right) \delta\left(\mu - 1\right) \tag{F.10}$$

Here δ represents the Dirac delta function, whose application to the transport correction makes the term forward peaked in the direction of travel to capture higher order anisotropies. Taking advantage of the relationship in Eq. F.9 and the approximation of the scattering kernel in Eq. F.10, the true Legendre polynomial scattering coefficients can be related to the modified coefficients in Eq. F.11

$$\begin{aligned}\Sigma_{s,\ell}\left(\mathbf{r},E'\rightarrow E\right) \approx \int_{-1}^{1} d\mu \sum_{\ell'=0}^{L}\frac{2\ell'+1}{2}\tilde{\Sigma}_{s,\ell'}\left(\mathbf{r},E'\rightarrow E\right)P_{\ell'}(\mu)P_{\ell}(\mu) \\ + \int_{-1}^{1} d\mu\, \Delta\Sigma_{tr}\left(\mathbf{r},E'\rightarrow E\right)\delta\left(\mu-1\right)P_{\ell}(\mu)\end{aligned} \tag{F.11}$$

For $0 \leq \ell \leq L$, Eq. F.11 can be simplified using the orthogonality of Legendre polynomials and the property $P_{\ell}(1) = 1$. This leads to the simplified relationship in Eq. F.12.

$$\Sigma_{s,\ell}\left(\mathbf{r},E'\rightarrow E\right) \approx \tilde{\Sigma}_{s,\ell}\left(\mathbf{r},E'\rightarrow E\right) + \Delta\Sigma_{tr}\left(\mathbf{r},E'\rightarrow E\right) \tag{F.12}$$

In order to capture the next order scattering after L, the transport correction term is set to capture scattering of order $L+1$ as shown in Eq. F.13.

$$\Delta\Sigma_{tr}\left(\mathbf{r},E'\rightarrow E\right) = \Sigma_{s,L+1}\left(\mathbf{r},E'\rightarrow E\right) \tag{F.13}$$

For isotropic in lab scattering with $L = 0$, the scattering kernel takes the form of Eq. F.14, following the form of Eq. F.10, where the transport correction term compensates for first order scattering in the direction of travel.

$$\begin{aligned}\Sigma_s(\mathbf{r},\mathbf{\Omega}'\rightarrow\mathbf{\Omega},E'\rightarrow E) \approx & \frac{1}{4\pi}\left[\Sigma_{s,0}(\mathbf{r},E'\rightarrow E) - \Sigma_{s,1}(\mathbf{r},E'\rightarrow E)\right] \\ & + \Sigma_{s,1}(\mathbf{r},E'\rightarrow E)\delta(\mathbf{\Omega}'\cdot\mathbf{\Omega}-1)\end{aligned} \tag{F.14}$$

Inserting this definition of the scattering kernel into Eq. F.7, the continuous energy and angle transport equation becomes:

$$\begin{aligned}
&\mathbf{\Omega}\cdot\nabla\psi(\mathbf{r},\mathbf{\Omega},E)+\Sigma_t(\mathbf{r},E)\psi(\mathbf{r},\mathbf{\Omega},E) = \\
&\qquad\frac{\chi(\mathbf{r},E)}{4\pi k}\int_0^\infty dE'\,\nu(\mathbf{r},E')\Sigma_f(\mathbf{r},E')\phi(\mathbf{r},E') \\
&\qquad+\int_0^\infty dE'\int_{4\pi} d\mathbf{\Omega}'\Bigg(\frac{1}{4\pi}\left[\Sigma_{s,0}(\mathbf{r},E'\to E)-\Sigma_{s,1}(\mathbf{r},E'\to E)\right] \\
&\qquad\qquad+\Sigma_{s,1}(\mathbf{r},E'\to E)\delta(\mathbf{\Omega}'\cdot\mathbf{\Omega}-1)\Bigg)\psi(\mathbf{r},\mathbf{\Omega}',E')
\end{aligned} \tag{F.15}$$

Taking advantage of the Dirac delta function in the transport correction term as well as the angular independence of the other scattering terms, this relationship can be simplified, as shown in Eq. F.16.

$$\begin{aligned}
&\mathbf{\Omega}\cdot\nabla\psi(\mathbf{r},\mathbf{\Omega},E)+\Sigma_t(\mathbf{r},E)\psi(\mathbf{r},\mathbf{\Omega},E)-\int_0^\infty dE'\,\Sigma_{s,1}(\mathbf{r},E'\to E)\psi(\mathbf{r},\mathbf{\Omega},E') = \\
&\qquad\frac{\chi(\mathbf{r},E)}{4\pi k}\int_0^\infty dE'\,\nu(\mathbf{r},E')\Sigma_f(\mathbf{r},E')\phi(\mathbf{r},E') \\
&\qquad+\frac{1}{4\pi}\int_0^\infty dE'\,\left(\Sigma_{s,0}(\mathbf{r},E'\to E)-\Sigma_{s,1}(\mathbf{r},E'\to E)\right)\phi(\mathbf{r},E')
\end{aligned} \tag{F.16}$$

A transport correction independent of outgoing neutron energy is defined in Eq. F.17. Similar to how the multi-group total cross-section is angle dependent, but the continuous energy total cross-section is independent of angle, the transport correction also becomes angular dependent.

$$\Delta\Sigma_{tr}(\mathbf{r},\mathbf{\Omega},E)=\frac{\int_0^\infty dE'\,\Sigma_{s,1}(\mathbf{r},E'\to E)\psi(\mathbf{r},\mathbf{\Omega},E')}{\psi(\mathbf{r},\mathbf{\Omega},E)} \tag{F.17}$$

However, similar to how the angular dependence of the total cross-section is ignored, the angular dependence of the transport correction is also ignored. In this thesis the same approximation is invoked. Although this approximation could be significant, the use of a first order correction for the transport correction already introduces some bias. Taking the transport correction to be angular independent, the transport correction is defined in terms of scalar fluxes in Eq. F.18.

$$\Delta\Sigma_{tr}(\mathbf{r},E) = \frac{\int_0^\infty dE'\,\Sigma_{s,1}(\mathbf{r},E'\to E)\phi(\mathbf{r},E')}{\phi(\mathbf{r},E)} \tag{F.18}$$

This leads to the transport equation shown in Eq. F.19

$$\begin{aligned}\boldsymbol{\Omega}\cdot\nabla\psi(\mathbf{r},\boldsymbol{\Omega},E)+\Sigma_{tr}(\mathbf{r},E)\psi(\mathbf{r},\boldsymbol{\Omega},E) = \\ \frac{\chi(\mathbf{r},E)}{4\pi k}\int_0^\infty dE'\,\nu(\mathbf{r},E')\Sigma_f(\mathbf{r},E')\phi(\mathbf{r},E') + \frac{1}{4\pi}\int_0^\infty dE'\,\tilde{\Sigma}_s(\mathbf{r},E'\to E)\phi(\mathbf{r},E')\end{aligned} \tag{F.19}$$

where the modified total cross section $\Sigma_{tr}(\mathbf{r},E)$ and modified scattering kernel $\tilde{\Sigma}_s(\mathbf{r},E'\to E)$ are defined in Eq. F.20 and Eq. F.21, respectively.

$$\Sigma_{tr}(\mathbf{r},E) = \Sigma_t(\mathbf{r},E) - \Delta\Sigma_{tr}(\mathbf{r},E) \tag{F.20}$$

$$\tilde{\Sigma}_s(\mathbf{r},E'\to E) = \Sigma_{s,0}(\mathbf{r},E'\to E) - \Delta\Sigma_{tr}(\mathbf{r},E)\delta(E'-E) \tag{F.21}$$

The modified total cross-section is often referred to as the *transport cross-section*. Note that the Dirac delta function enters the expression in Eq. F.21 since the transport correction term is now independent of the outgoing neutron energy, but it enters the scattering kernel which is dependent on outgoing neutron energy.

An equivalent multi-group form can also be derived in Eq. F.22

$$\boldsymbol{\Omega}\cdot\nabla\psi_g(\mathbf{r},\boldsymbol{\Omega})+\Sigma_{tr}(\mathbf{r})\psi_g(\mathbf{r},\boldsymbol{\Omega}) = \frac{1}{4\pi}\left(\frac{\chi_g(\mathbf{r})}{k}\sum_{g'=1}^{G}\nu_{g'}(\mathbf{r})\Sigma_f^{g'}(\mathbf{r})\phi_{g'}(\mathbf{r}) + \sum_{g'=1}^{G}\tilde{\Sigma}_s^{g'\to g}(\mathbf{r})\phi_{g'}(\mathbf{r})\right) \tag{F.22}$$

where the multi-group transport cross section $\Sigma_{tr}^g(\mathbf{r})$ and modified scattering kernel definitions are given in Eq. F.23 and Eq. F.24, respectively. Both are based on a multi-group transport correction term $\Delta\Sigma_{tr}^g(\mathbf{r})$ which is given in Eq. F.25.

$$\Sigma_{tr}^g(\mathbf{r}) = \Sigma_t^g(\mathbf{r}) - \Delta\Sigma_{tr}^g(\mathbf{r}) \tag{F.23}$$

$$\tilde{\Sigma}_s^{g'\to g}(\mathbf{r}) = \Sigma_s^{g'\to g}(\mathbf{r}) - \Delta\Sigma_{tr}^g(\mathbf{r})\delta_{g',g} \tag{F.24}$$

$$\Delta\Sigma_{tr}^g(\mathbf{r}) = \frac{\int_{E_g}^{E_{g-1}} dE \int_0^\infty dE'\, \Sigma_{s,1}(\mathbf{r}, E' \to E)\phi(\mathbf{r}, E')}{\int_{E_g}^{E_{g-1}} dE\, \phi(\mathbf{r}, E)} \tag{F.25}$$

In Eq. F.24, $\delta_{g',g}$ represents the Kronecker delta function. Its application to the transport correction term indicates that it is only applied to in-group scattering. The unmodified scattering kernel $\Sigma_s^{g'\to g}(\mathbf{r})$ represents the scattering kernel assuming isotropic scattering. This definition is often termed the flux-limited approximation [75].

In practice, the transport correction can easily be incorporated into codes that rely on isotropic scattering since it is equivalent to just modifying the underlying cross-section data. Therefore, from the perspective of designing a transport solver, the transport correction is just treated as a modification to the cross-section inputs.

F.4 Monte Carlo Cross-section Generation with OpenMC

Using direct Monte Carlo simulation of the BEAVRS benchmark with the OpenMC code, reaction rate tallies are generated for each unique material. These allow for the computation of multi-group cross-sections with the methodology discussed in Section F.1 using the `mgxs` package implemented by Boyd [1]. The Monte Carlo simulation used the JEFF-3.2 cross-section data at a temperature of 566.483K. 400 batches (300 inactive, 100 active) were simulated with 2×10^8 particles per batch to tally the 70 group cross-section library.

The flux-limited transport correction described in Section F.3 is applied to the cross-sections using anisotropic scattering rate tallies. Note that using these tallies to form the transport correction introduces approximation since the true tallies should involve angular fluxes rather than scalar fluxes. Since the anisotropic scattering rate tallies could vary significantly between core and reflector regions, the water is split into two materials for which cross-sections are independently formed: core water and outer reflector water. In addition, a third water material is also formed near the support plate / nozzle as the isotopic composition differs due to boron concentration. Instrument tubes are also scattered through the core, causing the problem to not be quadrant symmetric. Plots of the unique materials for which cross-sections are generated can be found in Appendix E.

The computed cross-sections were compared with CASMO-4 cross-sections and significant differences were found in the transport cross-section for water. In preliminary tests, the CASMO-4 multi-group cross-sections for core water were also able to much more accurately simulate the radial fission distribution. Therefore, instead of solely using the OpenMC `mgxs` cross-sections for core water which had an inaccurate transport correction, or solely using CASMO-4 cross-sections which are generated for more general problems, a new cross-section set was formed specifically for core water. Since CASMO-4 is a lattice physics code, it is designed to have accurate estimates of core water cross-sections. Therefore, the transport correction from CASMO-4 is used to modify the cross-sections formed by the OpenMC `mgxs` package. Specifically, the transport cross-section Σ_{tr}^{g} for group g is formed by computing

$$\Sigma_{tr}^{g} = \Sigma_{a}^{g} + \eta\left(\Sigma_{t}^{g} - \Sigma_{a}^{g}\right) \tag{F.26}$$

where Σ_{a}^{g} and Σ_{t}^{g} are the associated absorption and total cross-sections formed by `mgxs`, respectively. The factor η is computed by

$$\eta = \frac{\Sigma_{tr}^{g,\textbf{CASMO}} - \Sigma_{a}^{g,\textbf{CASMO}}}{\Sigma_{t}^{g,\textbf{CASMO}} - \Sigma_{a}^{g,\textbf{CASMO}}} \tag{F.27}$$

where the **CASMO** superscript denotes CASMO-4 cross-sections. It is important to note that this is only done for core water. All other materials use the cross-sections formed directly from the `mgxs` package in OpenMC. A comparison of η computed by OpenMC and by CASMO-4 for water is shown in Figure F-1.

In this appendix, the mechanics and theory behind multi-group cross-section generation were discussed. The focus was placed on Monte Carlo generation of cross-sections for full core simulation. In this process, it is important to highlight that while continuous energy cross-sections only depend on material properties, multi-group cross-sections are region-dependent due to dependence on the flux spectrum within each region. This effect

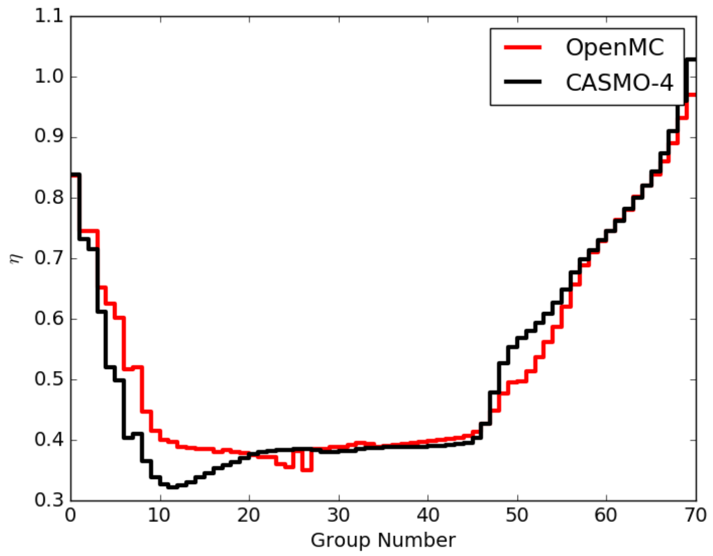


Figure F-1: A comparison of the η factor defined in Eq. F.27 for cross-sections generated with OpenMC and the CASMO-4 cross-sections for water in 70 energy groups.

can be quite significant. However, at large number of energy groups, the cross-sections more closely resemble the continuous energy cross-sections and the spatial dependence is diminished. This thesis concentrates on using a relatively large number of energy groups so that cross-sections can be treated as largely material dependent rather than having additional spatial dependence.

References

[1] W. R. D. BOYD, *Reactor Agnostic Multi-Group Cross Section Generation for Fine-Mesh Deterministic Neutron Transport Simulations*, PhD thesis, Massachusetts Institute of Technology, 2017.

[2] B. W. KELLEY and E. W. LARSEN, "A consistent 2D/1D approximation to the 3D neutron transport equation," *Nuclear Engineering and Design*, **295**, 568 (2015).

[3] Z. LIU, L. LIANG, C. LIANGZHI, and W. HONGCHUN, "A 2D/1D coupling method for transport calculation," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

[4] X. TANG, Q. LI, X. TU, W. WU, and W. K., "Efficient Procedure for Radial MOC and Axial SN coupled 3D Neutron Transport Calculation," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

[5] A. GRAHAM, B. COLLINS, and T. DOWNAR, "Improvement of the 2D/1D Method in MPACT Using the Subplane Scheme," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

[6] M. JARRETT, B. KOCHUNAS, E. LARSEN, and T. DOWNAR, "Progress in Characterizing 2D/1D Accuracy in MPACT," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

[7] W. WU, Q. LI, and K. WANG, "Verification of the 2D/1D Coupling 3D Transport Code TIGER with C5G7 Benchmarks," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

[8] M. HALSALL, "CACTUS, A Characteristics Solution to the Neutron Transport Equation in Complicated Geometries," United Kingdom Atomic Energy Establishment (1980).

[9] T. NEWTON, G. HOSKING, L. HUTTON, D. POWNEY, B. TURLAND, and T. SHUTTLEWORTH, "Developments within WIMS 10," *Proc. International Conference on the Physics of Reactors*, Interlaken, Switzerland, 2008.

[10] B. LINDLEY et al., "Developments within the WIMS Reactor Physics Code for Whole Core Calculations," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

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

[12] M. DAHAMANI, G. WU, R. ROY, and J. KOCLAS, "Development and Parallelization of the Three-Dimensional Characteristics Solver MCI of DRAGON," *Proc. PHYSOR*, Seoul, South Korea, 2002.

[13] C. RABITI, M. SMITH, Y. SIK, D. KAUSHIK, and G. PALMIOTTI, "Parallel method of characteristics on unstructured meshes for the UNIC code," *Proc. International Conference on the Physics of Reactors*, Interlaken, Switzerland, 2008.

[14] W. GROPP, E. LUSK, N. DOSS, and A. SKJELLUM, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel computing*, **22**, *6*, 789 (1996).

[15] M. SMITH, A. MARIN-LAFLECHE, W. YANG, D. KAUSHIK, and A. SIEGEL, "Method of Characteristics Development Targeting the High Performance Blue Gene/P Computer at Argonne National Laboratory,".

[16] A. MARIN-LAFLECHE, M. SMITH, and C. LEE, "PROTEUS-MOC: A 3D Deterministic Solver Incorporating 2D Method of Characteristics," *Proc. International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Sun Valley, ID, USA, 2013.

[17] 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*, **38**, 447 (2011).

[18] S. SHANER, G. GUNOW, B. FORGET, and K. SMITH, "Theoretical Analysis of Track Generation in 3D Method of Characteristics," *Proc. International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Nashville, TN, USA, 2015.

[19] B. KOCHUNAS, T. DOWNAR, and Z. LIU, "Parallel 3-D Method of Characteristics in MPACT," *Proc. International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Sun Valley, ID, USA, 2013.

[20] B. KOCHUNAS, *A Hybrid Parallel Algorithm for the 3-D Method of Characteristics Solution of the Boltzmann Transport Equation on High Performance Computing Clusters*, PhD thesis, University of Michigan, 2013.

[21] B. KOCHUNAS, B. COLLINS, S. STIMPSON, R. SALKO, D. JABAAY, A. GRAHAM, Y. LIU, K. S. KIM, W. WIESELQUIST, A. GODFREY, K. CLARNO, S. PALMTAG, T. DOWNAR, and J. GEHIN, "VERA Core Simulator Methodology for Pressurized Water Reactor Cycle Depletion," *Nuclear Science and Engineering*, **185**, *1*, 217 (2017).

[22] B. KOCHUNAS, T. DOWNAR, and Z. LIU, "Application of the SDD-CMFD Acceleration Method to Parallel 3-D MOC Transport," *Proc. PHYSOR*, Kyoto, Japan, 2014.

[23] T. TAKEDA and H. IKEDA, "3-D Neutron Transport Benchmarks," Organisation for Economic Co-operation and Development's Nuclear Energy Agency (1991).

[24] M. SMITH, E. LEWIS, and B. NA, *Benchmark on Deterministic Transport Calculations Without Spatial Homogenisation: MOX Fuel Assembly 3-D Extension Case*, Organisation of Economic Co-operation and Development - Nuclear Energy Agency (2005).

[25] D. SCIANNANDRONE, S. SANTANDREA, and R. SANCHEZ, "Optimized tracking strategies for step MOC calculations in extruded 3D axial geometries," *Annals of Nuclear Energy*, **87**, 49 (2016).

[26] P. ARCHIER, J. PALAU, S. SANTANDREA, and D. SCIANNANDRONE, "Validation of the Newly Implemented 3D TDT-MOC Solver of APOLLO3 Code on a Whole 3D SFR Heterogeneous Assembly," *Proc. PHYSOR*, Sun Valley, ID, USA, 2016.

[27] J. PALAU et al., "Recent Progress in the V&V of the New CEA APOLLO3 Code: Advanced SFR/LWR Assembly Calculations," *Proc. PHYSOR*, Sun Valley, ID, USA, 2016.

[28] L. GRAZIANO, S. SANTANDREA, D. SCIANNANDRONE, and I. ZMIJAREVIC, "Polynomial Characteristics Method for Neutron Transport in 3D extruded geometries," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

[29] S. SANTANDREA, L. GRAZIANO, and D. SCIANNANDRONE, "Optimized tracking strategies for step MOC calculations in extruded 3D axial geometries," *Annals of Nuclear Energy*, **113**, 194 (2018).

[30] A. GIHO, K. SAKAI, Y. IMAMURA, H. SAKURAGI, and K. MIYAWAKI, "Development of Axially Simplified Method of Characteristics in Three-Dimensional Geometry," *Journal of Nuclear Science and Technology*, **45**, *10*, 985 (2008).

[31] Y. KATO, T. ENDO, and A. YAMAMOTO, "Development of Legendre Expansion of Angular Flux Method for 3D MOC Calculation," *Proc. PHYSOR*, Kyoto, Japan, 2014.

[32] A. YAMAMOTO, A. GIHO, Y. KATO, and T. ENDO, "GENESIS: A Three-Dimensional Heterogeneous Transport Solver Based on the Legendre Polynomial Expansion of Angular Flux Method," *Nuclear Science and Engineering*, **186**, 1 (2017).

[33] A. F. HENRY, *Nuclear Reactor Analysis*, The MIT Press (1975).

[34] J. J. DUDERSTADT and L. J. HAMILTON, *Nuclear Reactor Analysis*, John Wiley & Sons (1976).

[35] J. J. DUDERSTADT and W. R. MARTIN, *Transport Theory,* John Wiley & Sons (1979).

[36] G. BELL, G. HANSEN, and H. SANDMEIER, "Multitable Treatments of Anisotropic Scattering in S N Multigroup Transport Calculations," *Nuclear Science and Engineering*, **28**, *3*, 376 (1967).

[37] A. HÉBERT, *Applied Reactor Physics*, Presses inter Polytechnique (2009).

[38] B. KOUCHUNAS, T. DOWNAR, S. MOHAMED, and J. THOMAS, "Improved Parallelization of the Modular Ray Tracing in the Method of Characteristics Code DeCART," *Proc. Joint Int'l Topical Meeting on Math. & Comp. and Supercomp. in Nucl. Appl.*, Monterey, CA, 2007.

[39] D. GASTON, B. FORGET, K. SMITH, and R. MARTINEAU, "Verification of MOCkingbird, an Unstructured-Mesh, Method of Characteristics Implementation Using the MOOSE Multiphysics Framework," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

[40] J. RHODES, K. SMITH, and D. LEE, "CASMO-5 Development and Applications," *Proc. ANS Topical Meeting on Reactor Physics (PHYSOR)*, p. 10–14, 2006.

[41] R. FERRER, J. RHODES, and K. SMITH, "Linear Source Approximation in CASMO5," *Proc. PHYSOR*, Knoxville, TN, USA, 2012.

[42] R. FERRER and J. RHODES, "A Linear Source Approximation Scheme for the Method of Characteristics," volume 77, p. 119–136, 1981.

[43] W. Boyd, S. Shaner, L. Li, B. Forget, and K. Smith, "The OpenMOC Method of Characteristics Neutral Particle Transport Code," *Annals of Nuclear Energy* (2014).

[44] D. M. BEAZLEY, "Automated Scientific Software Scripting with SWIG," *Future Generation Computer Systems*, **19**, *5*, 599 (2003).

[45] O. A. R. BOARD, "OpenMP Application Program Interface, Version 3.1," `http://www.openmp.org`, 2011.

[46] W. BOYD, S. SHANER, L. LI, B. FORGET, and K. SMITH, "The OpenMOC Method of Characteristics Neutral Particle Transport Code," *Annals of Nuclear Energy*, **68**, 43 (2014).

[47] N. J. HIGHAM, "The Accuracy of Floating Point Summation," *SIAM Journal on Scientific Computing,*, **14**, *4*, 783 (1993).

[48] M. HEARLIHY and N. SHAVIT, *The Art of Multiprocessor Programming,* Morgan Kaufmann Publishers (2008).

[49] G. GUNOW, J. TRAMM, B. FORGET, K. SMITH, and T. HE, "SimpleMOC – A performance Abstraction for 3D MOC," *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering* (2015).

[50] C. JOSEY, *Personal Communication* (2017).

[51] D. PATTERSON, T. ANDERSON, N. CARDWELL, R. FROMM, K. KEETON, C. KOZYRAKIS, R. THOMAS, and K. YELICK, "A Case for Intelligent RAM," *IEEE Micro*, **17**, *2*, 34 (1997).

[52] A. YAMAMOTO, M. TABUCHI, N. SUGIMURA, T. USHIO, and M. MORI, "Derivation of Optimum Polar Angle Quadrature Set for the Method of Characteristics Based on Approximation Error for the Bickley Function," *Journal of Nuclear Science and Technology*, **44**, *2*, 129 (2007).

[53] W. L. FILIPPONE, S. WOOLF, and R. J. LAVIGNE, "Particle Transport Calculations with the Method of Streaming Rays," *Nuclear Science and Engineering*, **77**, 119 (1981).

[54] J. ARVO and D. KIRK, *An Introduction to Ray Tracing*, Academic Press (1989).

[55] S. RUBIN and T. WHITTED, "A 3D representation for fast rendering of complex scenes," *Proc. SIGGRAPH*, p. 110–116, 1980.

[56] G. GUNOW, S. SHANER, B. FORGET, and K. SMITH, "Reducing 3D MOC Storage Requirements with Axial On-the-fly Ray Tracing," *Proc. PHYSOR 2016*, Sun Valley, ID, USA, May, 2016.

[57] G. GUNOW, S. SHANER, W. BOYD, B. FORGET, and K. SMITH, "Accuracy and Performance of 3D MOC for Full-Core PWR Problems," *Proc. Int'l Conf. on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.

[58] L. LAMPORT, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," *IEEE Transactions on Computers*, **C-29**, *9*, 690 (1979).

[59] M. TABUCHI, A. YAMAMOTO, T. ENDO, and N. SUGIMURA, "Convergence analysis of MOC inner iterations with large negative self-scattering cross-section," *Journal of Nuclear Science and Technology*, **50**, *5*, 493 (2013).

[60] M. TABUCHI, M. TATSUMI, A. YAMAMOTO, and T. ENDO, "Improvement of a Convergence Technique for MOC Calculation with Large Negative Self-scattering Cross-section," *Proc. PHYSOR*, Kyoto, Japan, September, 2014.

[61] B. BRADIE, *A Friendly Introduction to Numerical Analysis*, Pearson Prentice Hall, Upper Saddle River, New Jersey (2006).

[62] A. GRAHAM, *Nonnegative Matrices and Applicable Topics in Linear Algebra*, John Wiley & Sons (1987).

[63] M. EDENIUS, K. EKBERG, B. H. FORSSÉN, and D. KNOTT, "CASMO-4, A Fuel Assembly Burnup Program, User's Manual," *Studsvik0SOA-9501, Studsvik of America, Inc.* (1995).

[64] S. SHANER, G. GUNOW, B. FORGET, and K. SMITH, "Verification of the 3D Method of characteristics solver in OpenMOC," *Proc. PHYSOR 2016*, Sun Valley, ID, USA, May, 2016.

[65] W. BOYD, N. GIBSON, B. FORGET, and K. SMITH, "An Analysis of Condensation Errors in Multi-Group Cross Section Generation for Fine-Mesh Neutron Transport Calculations," *submitted to Annals of Nuclear Energy*.

[66] G. GIUDICELLI, K. SMITH, and B. FORGET, "Generalized equivalence methods for 3D multi-group neutron transport," *Annals of Nuclear Energy*, **112**, 9 (2018).

[67] Z. LIU, K. SMITH, B. FORGET, and J. ORTENSI, "Cumulative migration method for computing rigorous diffusion coefficients and transport cross sections from Monte Carlo," *Annals of Nuclear Energy*, **112**, 507 (2018).

[68] J. TRAMM, K. SMITH, B. FORGET, and A. SIEGEL, "ARRC: A random ray neutron transport code for nuclear reactor simulation," *Annals of Nuclear Energy*, **112**, 693 (2018).

[69] K. S. SMITH, "Nodal Method Storage Reduction by Non-linear Iteration," volume 44, 1983.

[70] C. HANSEN, *Numerical Methods of Reactor Analysis*, Academic Press (1964).

[71] K. S. SMITH and J. D. RHODES, "Full-Core, 2-D, LWR Core Calculations with CASMO-4E," *Proc. PHYSOR*, Seoul, South Korea, 2002.

[72] N. HORELIK, B. HERMAN, B. FORGET, and K. SMITH, "Benchmark for Evaluation and Validation of Reactor Simulations (BEAVRS), v1.0.1," *Proc. Int. Conf. Math. and Comp. Methods Applied to Nuc. Sci. & Eng.*, Sun Valley, Idaho, USA, 2013.

[73] R. MACFARLANE, "TRANSX 2: A Code for interfacing MATXS Cross-Section Libraries to Nuclear Transport Codes," Los Alamos National Laboratory (1993).

[74] R. MACFARLANE, "PSR-480/NJOY99.0: Code System for Producing Pointwise and Multigroup Neutron and Photon Cross Sections from ENDF/B Data," Los Alamos National Laboratory (2000).

[75] A. YAMAMOTO, Y. KITAMURA, and Y. YAMANE, "Simplified Treatments of Anisotropic Scattering in LWR Core Calculations," *Journal of Nuclear Science and Technology*, **45**, *3*, 217 (2008).