AN EXTENDED SLICE BALANCE APPROACH FOR SOLVING THE DISCRETE ORDINATES NEUTRAL PARTICLE TRANSPORT EQUATIONS ON THE NEXT GENERATION OF SUPER-COMPUTERS

A Dissertation

by

RICHARD MANUEL VEGA

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2019

Major Subject: Nuclear Engineering

ABSTRACT

The research presented in this document extends the slice balance approach (SBA) for solving the discrete ordinates neutral particle transport equations. The extended slice balance approach (ESBA) formulated here improves the accuracy of the underlying spatial discretization scheme in the presence of shadow-type discontinuities by exploiting the new concept of a sub-slice.

This research also derives and employs the linear discontinuous finite element (LDFE) spatial discretization scheme within the ESBA framework. Current codes utilizing the SBA rely on low-accuracy discretization schemes, because the geometric information required for higher accuracy schemes has been seen as too voluminous to store and too computationally expensive to re-calculate each time it is needed. Here we show that the judicious use of modern hardware such as the graphics processing unit (GPU) can speed the re-calculation of geometric quantities by factors of a few hundred compared to a single core, raising the possibility that more accurate SBA and ESBA methods may become practical if such hardware is employed.

The re-definition of a slice such that no slice may straddle any arbitrarily placed cut plane parallel to the discrete ordinate, leads to the region in between adjacent cut planes being completely independent of any other such region during a transport "sweep." This provides the ability to divide the mesh into independent regions, resulting in consequences that lead to two new parallel sweep strategies introduced in this document.

When considering the LDFE discretization scheme, the ESBA is found to reduce the absolute error for smooth solutions and increase the convergence rate for discontinuous solutions compared to the SBA, which similarly reduces the error and increases the convergence rate over the traditional cell balance approach (CBA). The two parallelization strategies made possible by the ESBA exhibit weak-scaling results similar to those obtained with a simple volume-decomposed parallel transport sweep for both the SBA and CBA. The acceleration of the slice and sub-slice formation process using GPUs is found to exhibit speedups of up to 400 times when compared to a single core of the host CPU.

ACKNOWLEDGEMENTS

No one completes a PhD dissertation without help and encouragement from many people along the way, and I am certainly no exception. The number of people to which I owe deep gratitude makes writing this section of this document exceptionally challenging, if for no other reason than it is difficult to decide whom to thank second. Choosing whom to thank first however, that task is easy. I would like to thank Dr. Marvin Adams for his guidance, patience, insight, and above all his generosity. This generosity was demonstrated to me as an undergraduate student when I was awarded the Adams' Family Scholarship, which quite literally allowed me to continue my education. It was demonstrated again as I was finishing up my dissertation with a personal scholarship when I was experiencing difficult financial circumstances. It is also true that his generosity takes forms other than money. He is gracious with his time when it is obvious he has none to spare and I would like to thank him for taking an interest in me as an undergraduate six years ago. He is the best professor I have had the privilege of learning from and the best advisor I could have asked for.

Next, I would like to thank my manager and mentors at Sandia National Laboratories where I have been a year-round intern for the past five years, Dr. Kenneth Reil, Dr. Russell Depriest, and Dr. Edward Parma. Under your guidance I have had the best training anyone could ask for to prepare myself for life after graduate school. The work that I have performed during this time gave me a different perspective than most other graduate students, and the experience has been truly vital to my success. I look forward to pursuing my post-graduate career among all of you, and I intend to make your investment in me over these past five years pay off many times over. Thank you for your patience during these last few months.

I would also like to thank the Department of Energy (DOE) National Nuclear Security Administration (NSSA) Stewardship Science Graduate Fellowship (SSGF), and all of the people who make this program possible, and in particular Lucille Kilmer, Kris Moran, and Michelle King. I undoubtedly would not be finishing my PhD anytime soon without all of the financial support and experiences that this program has provided me. It has been an amazing honor to meet all of the fellows over the last four years, and take part in all of the program reviews. The program has been indispensable to my graduate studies, and I cannot thank the Krell Institute and their employees enough for all of the hard work they do to keep the program running smoothly.

I would like to thank all of my professors at Texas A&M University, and in particular my committee members Dr. Nancy Amato, Dr. Jim Morel, and Dr. Jean Ragusa. I would like to thank you not only for the valuable classes that you have taught and the helpful advice you have given along the way, but also for reading this rather lengthy document. I would also like to thank Marna Stepan for guiding me through the administrative bureaucracy that I am sure all universities are host to. You were a light in the dark during my first year at Texas A&M University where I felt like a very small fish in the ocean. I truly appreciate everything each and every one of you has done to help me along the way.

I would like to thank my family. I know there have been bumps in the road, and I am sure I placed some of those bumps there myself, but I could not be where I am or who I am today without you. To my mother, you taught me to be caring and kind and to help those that I could help, and while I know you wish you could have done more, I don't think you will ever know how much you really did. To my dad, you picked me up when I hit rock bottom. Without your help, I honestly don't think I would be alive today, and I certainly wouldn't ever be Dr. Richard Vega.

Finally, I would like to thank my ex-wife Heather. You stayed with me through eight years of my decade in school, through both of my Bachelor's degrees, and throughout most of graduate school. You cooked, cleaned, and raised our two fur-sons (dogs), with minimal help, all while working a full-time job and completing your Bachelor's and Master's degrees as well. You gave me the best eight years of my life, and it was probably at least seven years longer than I deserved. I just want you to know how much I appreciated it. I didn't always show it, but you made me happier than I had ever been before, and probably ever will be again. Thank you.

CONTRIBUTORS AND FUNDING SOURCES

This work was supported by a dissertation committee consisting of Professors Marvin Adams, Jean Ragusa, and Jim Morel in the Nuclear Engineering department at Texas A&M University as well as Professor Nancy Amato in the Computer Science and Engineering department at Texas A&M University. The department head of the Nuclear Engineering department is Yassin Hassan. All other work conducted for the dissertation was completed by the student independently.

Graduate study was supported by a fellowship from Texas A&M University and a dissertation research fellowship from the Stewardship Science Graduate Fellowship awarded by the National Nuclear Science Administration (NNSA) within the United States Department of Energy (DOE).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1. INTRODUCTION

This chapter will provide necessary background material in the area of particle transport, present the motivations and objectives for this research, and provide a preview of the chapters to come. We begin by reviewing the fundamental equation at the heart of nuclear engineering and particle transport. We also describe the two common approaches to solving this equation, and briefly discuss their advantages and disadvantages. We discuss the need for modern particle transport codes that are able to run on the next generation of super-computers, and what capabilities should be included in such codes.

1.1 Background

Many fields in science and engineering can point to a single equation, or set of equations, that in theory could be used for predicting the outcomes of experiments and improving the capability for innovative design. Fluid mechanics has the Navier-Stokes equations, quantum mechanics has the Schrödinger equation, and particle transport has the Boltzmann equation. These three equations have several things in common; they result from a statement of conservation, approximations are often made to make them more manageable, and even after many approximations are made, they are still incredibly difficult to solve for any practical application. The approximations typically made to the Boltzmann equation in the study of neutral particle transport and nuclear reactor physics are listed below.[1]

- Particle motion between interactions is described by classical mechanics.
- Particles do not interact with each other, and only interact with the atoms and nuclei of the material through which they pass.

- Particles are not affected by external forces such as gravitational and electromagnetic forces.
- The thermal motion of the background atoms causes them to move isotropically, allowing particle interaction cross sections to be direction independent.

When these approximations are valid, the Boltzmann equation can be re-written in a form known as the linear Boltzmann equation, also known as the transport equation, which is an appropriate starting point for many studies of neutral particle transport:

$$\left(\frac{1}{v(E)}\frac{\partial}{\partial t} + \mathbf{\Omega}\cdot\nabla + \sigma_t\left(\mathbf{r},E,t\right)\right)\psi\left(\mathbf{r},\mathbf{\Omega},E,t\right) =$$

$$\iint_{4\pi}\int_0^\infty \sigma_s\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E,t\right)\psi\left(\mathbf{r},\mathbf{\Omega}',E',t\right)dE'd\Omega' + q\left(\mathbf{r},\mathbf{\Omega},E,t\right)\ , \quad (1.1)$$

where

$$\mathbf{r} = \text{spatial coordinate vector (cm)}\ ,$$

$$\mathbf{\Omega} = \text{particle unit directional vector}\ ,$$

$$E = \text{particle energy (MeV)}\ ,$$

$$t = \text{time (s)}\ ,$$

$$v\left(E\right) = \text{particle speed}\left(\frac{\text{cm}}{\text{s}}\right)\ ,$$

$$\psi\left(\mathbf{r},\mathbf{\Omega},E,t\right) = \text{particle angular flux}\left(\frac{\text{particles}}{\text{MeV}\cdot\text{ster}\cdot\text{cm}^2\cdot\text{s}}\right)\ ,$$

$$\sigma_t\left(\mathbf{r},E,t\right) = \text{total interaction cross section}\left(\frac{1}{\text{cm}}\right)\ ,$$

$$\sigma_s\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E,t\right) = \text{scattering cross section}\left(\frac{1}{\text{cm}\cdot\text{MeV}\cdot\text{ster}}\right)\ ,$$

and

$$q\left(\mathbf{r},\mathbf{\Omega},E,t\right) = \text{particle source rate density}\left(\frac{\text{particles}}{\text{MeV}\cdot\text{ster}\cdot\text{cm}^3\cdot\text{s}}\right) .$$

Equation 1.1 is an integro-partial differential equation for the angular flux, which has a seven-dimensional phase space comprising 3 parameters to define a point in space, two parameters to define a direction of flight, 1 parameter to specify particle energy, and 1 parameter to specify time. Another useful quantity is the scalar flux defined as

$$\phi\left(\mathbf{r},E,t\right) = \iint_{4\pi} \psi\left(\mathbf{r},\mathbf{\Omega},E,t\right) d\Omega .$$

This quantity is particularly important because it is the scalar flux that determines reaction rates, and it is reaction rates that are usually measured experimentally and sought for design purposes. The reaction rate in a given volume V over a specified time interval t_1 to t_2 is given by

$$R = \int_{t_1}^{t_2} \iiint_V \int_0^\infty \sigma_R\left(\mathbf{r},E,t\right)\phi\left(\mathbf{r},E,t\right) dE\, d^3r\, dt , \tag{1.2}$$

where $\sigma_R\left(\mathbf{r},E,t\right)$ is the cross section for the reaction of interest.

The transport equation represents the conservation of particles within every phase space volume. The left hand side contains the time rate of change of the particle number density, the loss rate due to leakage, and the loss rate due to particle interactions with the background material. The right hand side contains the gain rate due to in-scatter and external sources. The angular flux represents the expected value of the distribution of particles in space, direction, and energy as a function of time. Since particle transport is an inherently stochastic process, this distribution will have statistical noise associated with it. If the number density of particles is low, this statistical noise will be significant, and the angular flux in reality may stray far from this average distribution. For many applications however, the number density of particles is so large that there is very little deviation from this average distribution.

There exists a unique solution to equation 1.1 given appropriate initial and boundary conditions.[2] The research presented here is focused on finding these solutions as quickly, efficiently, and accurately as possible, while recognizing that these are often competing motivations. Initial conditions take the form

$$\psi\left(\mathbf{r},\boldsymbol{\Omega},E,t_0\right)=\psi_{\text{initial}}\left(\mathbf{r},\boldsymbol{\Omega},E\right)\ , \tag{1.3}$$

where $\psi_{\text{initial}}\left(\mathbf{r},\boldsymbol{\Omega},E\right)$ is known. Several types of boundary conditions can lead to a well-posed problem including an explicitly specified incident flux, specular or diffuse reflection, and periodic or albedo boundaries conditions. These can be written as

$$\psi\left(\mathbf{r},\boldsymbol{\Omega},E,t\right)=\psi_{\text{inc}}\left(\mathbf{r},\boldsymbol{\Omega},E,t\right)\ \text{for all } \boldsymbol{\Omega}:\mathbf{n}\left(\mathbf{r}\right)\cdot\boldsymbol{\Omega}<0 \text{ for all } \mathbf{r}\in\partial V_{\text{inc}}\ , \tag{1.4}$$

$$\psi\left(\mathbf{r},\boldsymbol{\Omega},E,t\right)=\psi\left(\mathbf{r},\boldsymbol{\Omega}_{\text{refl}},E,t\right)\ \text{for all } \boldsymbol{\Omega}:\mathbf{n}\left(\mathbf{r}\right)\cdot\boldsymbol{\Omega}<0 \text{ for all } \mathbf{r}\in\partial V_{\text{spec}}\ , \tag{1.5}$$

$$\psi\left(\mathbf{r},\boldsymbol{\Omega},E,t\right)=\frac{1}{\pi}\iint_{\boldsymbol{\Omega}':\mathbf{n}(\mathbf{r})\cdot\boldsymbol{\Omega}'>0}\mathbf{n}\left(\mathbf{r}\right)\cdot\boldsymbol{\Omega}'\,\psi\left(\mathbf{r},\boldsymbol{\Omega}',E,t\right)\,d\Omega' \\ \text{for all } \boldsymbol{\Omega}:\mathbf{n}\left(\mathbf{r}\right)\cdot\boldsymbol{\Omega}<0 \text{ for all } \mathbf{r}\in\partial V_{\text{diff}}\ , \tag{1.6}$$

$$\psi\left(\mathbf{r},\boldsymbol{\Omega},E,t\right)=\psi\left(\mathbf{r}+\mathbf{d}\left(\mathbf{r}\right),\boldsymbol{\Omega},E,t\right) \\ \text{for all } \boldsymbol{\Omega}:\mathbf{n}\left(\mathbf{r}\right)\cdot\boldsymbol{\Omega}<0 \text{ for all } \mathbf{r}\in\partial V_{\text{per}}\ , \tag{1.7}$$

and

$$\psi\left(\mathbf{r}, \boldsymbol{\Omega}, E, t\right) = \iint_{\boldsymbol{\Omega}' : \mathbf{n}(\mathbf{r}) \cdot \boldsymbol{\Omega}' > 0} \alpha\left(\mathbf{r}, E, \boldsymbol{\Omega}' \rightarrow \boldsymbol{\Omega}, t\right) \psi\left(\mathbf{r}, \boldsymbol{\Omega}', E, t\right) \, d\Omega'$$

$$\text{for all } \boldsymbol{\Omega} : \mathbf{n}\left(\mathbf{r}\right) \cdot \boldsymbol{\Omega} < 0 \text{ for all } \mathbf{r} \in \partial V_{\text{albedo}} \,, \quad (1.8)$$

respectively, where $\mathbf{n}\left(\mathbf{r}\right)$ is the outward pointing unit normal vector, $\psi_{\text{inc}}\left(\mathbf{r}, \boldsymbol{\Omega}, E, t\right)$ is known, $\boldsymbol{\Omega}_{\text{refl}}$ is the specular reflection of $\boldsymbol{\Omega}$, $\mathbf{d}\left(\mathbf{r}\right)$ is a translation vector, α is known as the albedo function, and ∂V_{inc}, ∂V_{spec}, ∂V_{diff}, ∂V_{per}, and $\partial V_{\text{albedo}}$ are the regions of the domain boundary ∂V, on which each condition is specified.

Analytic solutions to the transport equation are rare except for the simplest of problems. Such problems typically assume that the angular flux is energy independent, time independent, and only spatially dependent in one dimension. Of course, in reality such simplified problems either do not exist or are of little practical interest, and numerical solutions to the transport equation are often sought. These numerical solutions can be obtained via two distinct methodologies, deterministic and stochastic. The research presented here is an example of the former, and Chapter 2 will review some of the most common deterministic methods.

Deterministic methods attempt to find a solution to equation 1.1 by discretization or functional expansion of the angular flux in order to replace the original integro-partial differential equation with a system of algebraic equations. To discretize the angular flux phase space, a spatial mesh of the problem domain is produced consisting of non-overlapping cells. The energy domain is broken up into sub-domains, the particle direction can be discretized by either collocation or functional expansion, and the resulting discretized problem is solved at specified time intervals. To illustrate the difficulty imposed by the large phase space of the angular flux, imagine if the three spatial parameters, two directional parameters, and the particle energy were discretized into 100 bins each. The system of equations to be solved at each time step in the solution would contain 10^{12} unknowns. This large phase makes it impossible to store the entire solution in the memory of a single computer for even moderate resolutions of each of the seven independent variables.

On the bright side, if the complete angular flux solution can be obtained, a wealth of information becomes immediately available. The energy and angular distributions of the particles can be obtained at various locations and times, reaction rates can be determined in any spatial region of the domain, and these reaction rates can be used as inputs to other codes such as volumetric heat generation rates for a heat transfer calculation, or transmutation rates for an isotopic depletion calculation. This makes deterministic methods attractive for multi-physics applications where solutions are sought for multiple coupled partial differential equations simultaneously, perhaps even sharing the same spatial mesh. Such applications arise in the study of radiation hydrodynamics and nuclear reactor burnup calculations.

Of course, deterministic methods are not without their disadvantages. The most significant of these is related to the discretization of the energy variable, especially for neutron transport. The neutron interaction cross sections for many common nuclides have extremely complex energy dependence. In addition, it is not atypical for neutron energies to span eight orders of magnitude in many cases of interest. This makes capturing the complex energy dependence of the neutron cross sections with a reasonable number of energy groups impossible. The multi-group method used in deterministic approaches uses weighted cross sections in each group, and the calculation of the weighted cross sections to produce accurate results can be extremely onerous, and has itself been the subject of many dissertations.

The discretization of the direction variable is also a source of error, and this error can be challenging to quantify. To see why this is, consider an angular discretization method based on collocation where the transport equation is solved for a particular set of directions. In this discretization scheme, the true physics being represented is a physics in which particles can only travel in a discrete set of directions. This causes a very specific type of error known as a "ray effect," where the solution appears to have streaks along the discrete directions, most prominently in the vicinity of localized sources. While scattering cross sections could be computed for scattering from any direction to any other direction, it is more common to approximate the angular dependence of scattering cross sections with a truncated Legendre polynomial expansion.

Stochastic methods take a fundamentally different approach akin to a direct numerical simulation of reality. In a stochastic method, individual particles are simulated from birth at a source location, to absorption or leakage from the problem domain. Such simulations rely on pseudo-random numbers in order to sample from the probability distributions for such things as the distance to the next interaction, the interaction type, and the energy and direction of the particle at birth or after each scattering interaction. Interactions of interest are tallied, and as the number of simulated particles approaches the number of particles in the actual system, the results of these tallies become more accurate.

For the most part, stochastic transport codes are easier to setup and run than deterministic codes. Geometric models can be set up more easily and with higher fidelity because the only geometric information needed when a particle streams through a cell is the distance to the cell boundary. In addition, continuous energy cross sections can be used rather than multi-group cross sections. Continuous energy cross sections are easier to produce and are more accurate than multi-group cross sections, and this leads to a significant advantage in the treatment of the energy variable for stochastic methods. Finally, the directional dependence of scattering cross sections can be accurately represented, and particles can stream in the continuum of directions on the unit sphere as in reality.

Stochastic methods can be highly efficient when used to calculate limited information rather than the complete angular flux solution. Such limited information may be the dose rate at a physical location in space, the eigenvalue of a reactor core, or some other single quantity of interest (QoI). Consider the calculation of a detector response for instance. In such a calculation, a fine spatial mesh is unnecessary. Particles can be simulated starting from the source, and tallied if they interact in the detector volume. The geometric model only needs to delineate regions of different materials in the domain, significantly easing memory demands. In addition, source biasing techniques and importance sampling can be used to successfully steer more particles towards interacting in the detector, decreasing the time to solution.

This simulation of reality can seem quite intuitive. Unfortunately, the number of particles that can be simulated on even the most powerful super-computers in any reasonable amount of time, is likely to be several orders of magnitude less than the number of particles in the real system[1], leading to statistical errors in the numerical solution. The primary disadvantage associated with stochastic methods is this statistical error, and in particular the rate at which this error decreases as the number of particles simulated increases. This convergence rate is proportional to $N^{-1/2}$, where N is the number of particles simulated. To illustrate how slow this convergence rate is, note that in order to decrease the statistical error in any QoI by one order of magnitude, the number of particles simulated would have to increase by a factor of 100, and thus so would the computational resources.

In addition, the statistical error corresponding to a given tally is dependent upon the number of particles that contribute to it. One could imagine a very fine spatial mesh where the scalar flux is desired in each spatial cell, and interactions are then binned according to neutron energy at the time of interaction. Such a scenario is common in nuclear reactor burnup calculations. In this scenario, the finer the spatial and energy meshes, the fewer particles that will be tallied in each spatial and energy bin pair, and this will lead to high statistical error in each tally. While it is true that a finer spatial mesh increases the time to solution for deterministic methods as well, the convergence rate associated with the spatial discretization error is generally proportional to h^p, where h is a measure of the size of the cells in the mesh, and p is dependent upon the problem and spatial discretization method.

In pointing out the advantages and disadvantages of each type of method, it should be noted that this section is not intended to make a definitive case for either one over the other. If there were such a definitive case to be made, it is unlikely that both methods would have survived over the past half-century of independent research. Instead, the two approaches are complimentary, and sometimes one can be used to mitigate the deficiencies of the other. For instance, stochastic methods have been used to compute multi-group cross sections for deterministic methods[3], and deterministic methods have been used to calculate weight windows for importance sampling in stochastic methods.[4] In addition, which method is better is highly problem dependent. Stochastic methods excel when a single QoI is desired, and a significant fraction of the particles simulated can be made to contribute to this QoI, making them ideal for eigenvalue and detector response problems. Isotopic depletion problems on the other hand, may require the storage of cross sections for hundreds of nuclides, and the storage requirements for continuous energy cross sections for this many nuclides can become prohibitive, favoring less accurate multi-group cross sections instead. Table 1.1 lists the advantages and disadvantages mentioned above, with the disclaimer that a thorough review of the two approaches could not possibly be condensed into so few pages or a table of this size.

Table 1.1: Advantages and disadvantages of deterministic and stochastic methods.

Deterministic	
Advantages	**Disadvantages**
· A single simulation provides the necessary information to calculate any QoI	· Complex energy dependence of interaction cross sections cannot be accurately represented
· Well suited for multi-physics applications	· Direction discretization causes ray effects
· Multi-group cross sections reduce computer memory demands	· Angular dependence of scattering cross sections must be approximated, sometimes crudely
Stochastic	
Advantages	**Disadvantages**
· Can use more accurate continuous energy cross sections	· Statistical error decreases slowly
· Can more accurately treat scattering cross section angular dependence	· Continuous energy cross sections increase computer memory demands
· Can more easily handle complex geometries	· Not well suited for computing the complete angular flux solution on a fine mesh

1.2 Motivation and Objectives

A primary objective when developing a new method or code to solve the transport equation numerically, should be to implement it in such a way that it is able to run efficiently on modern super-computers. This means writing code that can run on many computers simultaneously, while minimizing idle time and communication time. Such super-computers are becoming increasingly heterogeneous, containing different types of processing units, and hence a brief review of such computer architectures is warranted. This review will include a short historical perspective for why such architectures have been chosen, and a glimpse into the current state of high performance computing (HPC) systems around the world.

Until the later part of the last century, computational physicists relied primarily on the computer science and engineering fields to produce faster central processing units (CPUs) in order to decrease the run times of their codes. This dependence was not unwarranted, given the impressive staying power of Moore's Law, which stated in 1965 that the number of transistors on a standard CPU would continue to double every two years for at least the next decade, and has in fact continued to present day. This led to a misconception that the CPU clock speed and hence overall performance would double as well, which stopped being true in 2005. The explanation for this is that as the clock speed and number of transistors increases, more heat is generated, requiring more sophisticated cooling systems. In addition, an increase in clock speed implied an increase in voltage (V) since 2005 when Dennard scaling began to stall due to current leakage, and the power consumption is proportional to V^3. Thus, clock speeds could not continue to rise indefinitely without causing the CPU to melt or consume unreasonable amounts of power.

The solution to these limitations was the multi-core processor. The idea is quite simple; lowering the voltage to 70% of its original value, which equates to a similar decrease in performance, the power consumption drops to 34% of its original value given the cubic relationship between power and voltage. This means that if instead there were two cores operating at 70% voltages, then you could conceivably get $2\times70\% = 140\%$ of the compute power of a single CPU operating at 100%, with $2\times34\% = 68\%$ of the power consumption. In addition, the lower clock speeds will result in less heat generation and ensure longer core lifetimes. It thus became apparent that there was no real need to increase the clock speed in favor of simply including more cores on each CPU. The number of transistors per chip, maximum clock speed, and thermal design power for CPUs since 1970 is shown in Figure 1.1.

While the transistor count continued its doubling since 2005, the clock speed and thermal power did not. If there were a fourth curve on Figure 1.1 indicating the number of cores, it would be constant at one until roughly 2005, and would then quickly rise to 4, 8, 12, and eventually into the 60's for the Intel Xeon Phi processor. This had profound effects for the theoretical peak performance of the standard CPU, but unfortunately code written to run on a single core does not magically run on multiple cores when they are available. To a programmer with no parallel programming experience, this meant that code written two years ago did not simply run twice as fast as it did then, as had been the case in the past. Parallel programming, which had been relegated to programmers working on computer clusters and super-computers, was now a useful skill to all programmers.

This trend of increasing the core count while capping the clock speed and power consumption was taken further in the graphics processing unit (GPU). Figure 1.2 shows the difference in architecture between a typical multi-core CPU and a GPU. The arithmetic-logic unit (ALU) in this figure is the equivalent of a core. The GPU was designed to take advantage of the concurrency of graphics processing, where each pixel on a screen could be rendered independently. This is an example of what has been termed single instruction, multiple data (SIMD) in Flynn's taxonomy of computer architectures[5], and is also known as embarrassingly parallel because no communication between the different tasks is required. With the advent of general purpose GPU (GPGPU) programming, these devices became useful for a wider variety of applications including scientific computing.

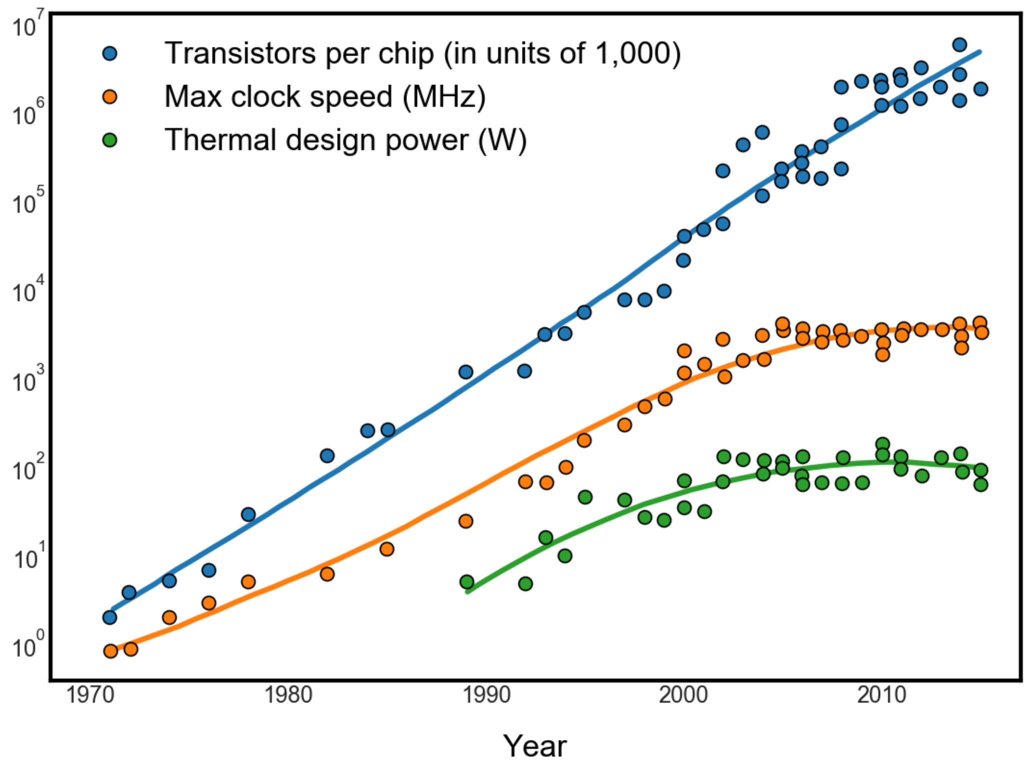


Figure 1.1: Transistors per chip, max clock speed, and thermal design power for CPUs since 1970, re-printed from the Economist.[6]

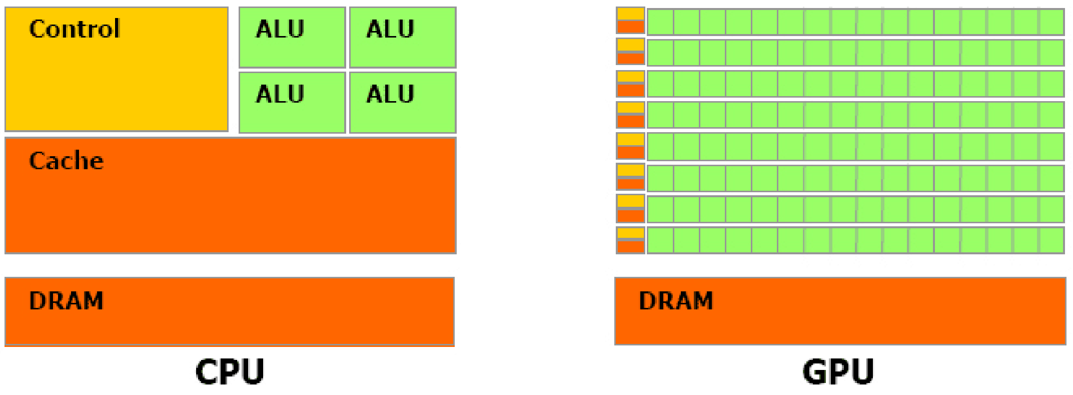


Figure 1.2: Comparison of CPU and GPU architectures.

While interest in GPGPU programming among the scientific community has increased, such specialized computer architectures have introduced more complexity into algorithm design. While the GPU may have thousands of cores, it also typically has a much smaller cache and significantly less memory per core than a multi-core CPU. This causes many applications to become more easily memory bound. There is also the issue of getting data to and from the GPU, which is usually carried via PCIe bus. All of these issues make the GPU ideal for some tasks, and less than ideal for others. More responsibility lies on the programmer to make the implementation of numerical methods more closely resemble the embarrassingly parallel process of pixel rendering in order to get the most out of the GPU. For applications where this is possible, significant performance gains can be achieved over the CPU as shown by the theoretical peak performance plotted in Figure 1.3.

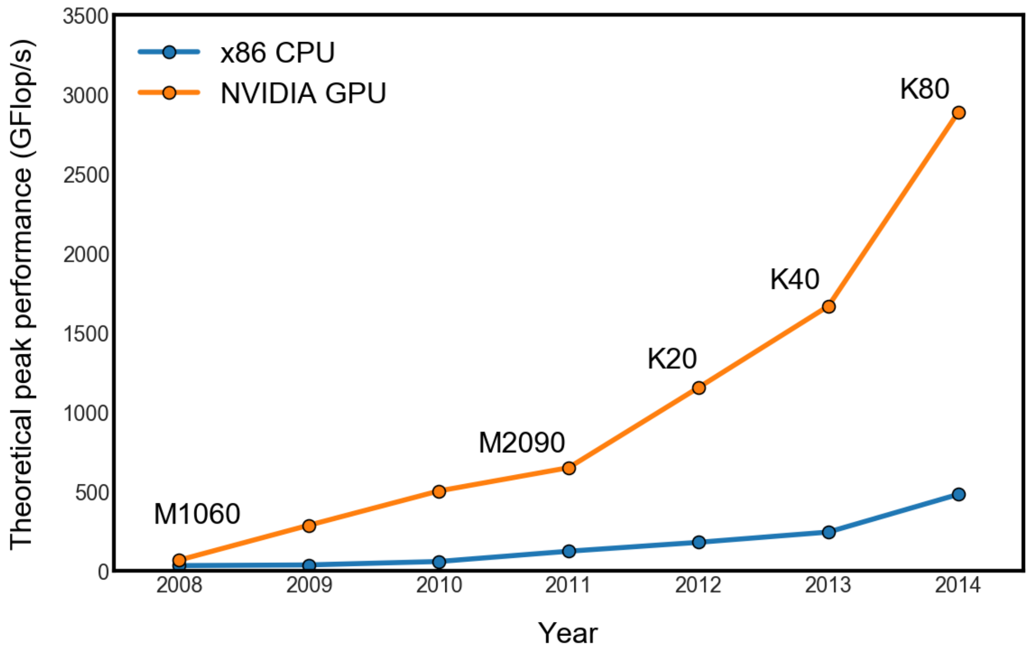


Figure 1.3: Comparison of CPU and GPU theoretical peak performance, re-printed from NVIDIA.[7]

Before multi-core CPUs and GPUs became common, parallel computing was an active field of research taking place on clusters and super-computers world-wide. While a multi-core CPU or a GPU incorporates shared memory, super-computers were usually distributed memory machines where hundreds or thousands of CPUs did not have access to the same memory, and instead had to send messages to communicate with one another. When the clock speed limit of a single CPU was realized, these super-computers became collections of nodes, each of which comprising one or more multi-core CPUs. Cores shared a memory bank with other cores on the same node, but had to send messages to cores on other nodes. More recently, super-computers have also taken advantage of the GPU, attaching a number of these devices to each node. Table 1.2 is a list of the world's top 10 super-computers as of June 2018 when ranked by maximal performance using the LINPACK benchmark (R_{max}), which is essentially a scalable dense matrix inversion problem.

Five of the top 10 super-computers in Table 1.2 incorporate at least one GPU on each node. The second ranked super-computer uses processing elements that are commonly referred to as many-core CPUs, such as the 260 core Sunway SW26010, blurring the lines between the CPU and GPU classification. Two computers on this list were recently procured by the United States Department of Energy (US DOE), Summit and Sierra. These two super-computers have multiple GPUs on each node, and have significantly higher performance than their predecessor Titan, while consuming roughly the same amount of power.[7] The term "next generation super-computers" used throughout this dissertation is specifically targeting these two machines. Ideally, codes to solve the transport equation should be flexible and be able to run efficiently on any of the machines listed in Table 1.2, however the heterogeneity of these super-computers often require implementations that target a specific architecture.

Table 1.2: List of the world's top 10 super-computers as of June 2018, re-printed from Top 500.[8]

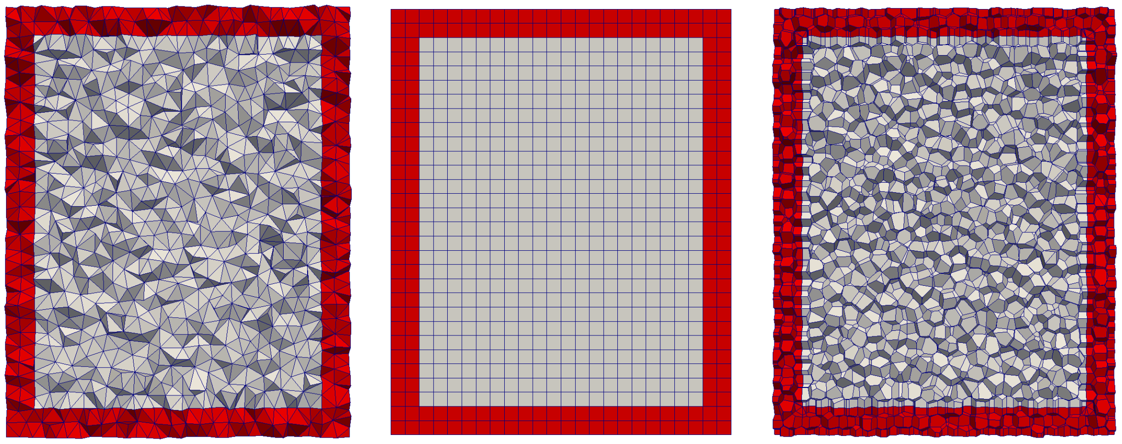
Name (location)	R_{max} (PFlop/s)	Power (MW)	Node description
Summit (USA)	122.3	8.8	IBM POWER9 22C 3.07 GHz, Dual-rail Mellanox EDR Infiniband interconnect, NVIDIA V100
Sunway TaihuLight (China)	93.0	15.4	Sunway SW26010 260C 1.45 GHz, Sunway interconnect
Sierra (USA)	71.6	5.2	IBM POWER9 22C 3.07 GHz, Dual-rail Mellanox EDR Infiniband interconnect, NVIDIA V100
Tianhe-2A (China)	61.4	18.5	Intel Xeon E5-2692v2 12C 2.2 GHz, TH Express-2 interconnect
AI Bridging Cloud Infrastructure (Japan)	19.9	1.6	Intel Xeon Gold 6148 20C 2.4GHz, Infiniband EDR interconnect, NVIDIA V100
Piz Daint (Switzerland)	19.6	2.3	Intel Xeon E5-2690v3 12C 2.6 GHz, Aries interconnect , NVIDIA P100
Titan (USA)	17.6	8.2	AMD Opteron 6274 16C 2.2 GHz, Cray Gemini interconnect, NVIDIA K20x
Sequoia (USA)	17.2	7.9	Power BQC 16C 1.6 GHz, Custom interconnect
Trinity (USA)	14.1	3.8	Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect
Cori (USA)	14.0	3.9	Intel Xeon Phi 7250 68C 1.4 GHz, Aries interconnect

In addition to the ability to efficiently run on the next generation of supercomputers, modern deterministic particle transport codes should have a number of other desirable properties. One such property is the ability to handle arbitrary polyhedral spatial meshes. Indeed, when Grove first introduced the Slice Balance Approach (SBA) [9], which will be discussed in detail in Chapter 2 and which this research builds upon, it was his view that one of the most useful properties of that method was its ability to extend planar spatial differencing schemes to arbitrary polyhedral meshes.

Unstructured spatial meshes used to model complex geometries are generally categorized by the shape of their cells. The simplest cell shape is the tetrahedron, having 4 points and four triangular faces. Because of this simplicity, tetrahedral meshes are able to capture extraordinarily complicated geometric features quite well. The next simplest cell shape is the hexahedron, comprised of 8 points and 6 quadrilateral faces. Modeling complex geometric features with a hexahedral mesh can be very difficult, especially if hanging nodes are not allowed. Of course, if the numerical method being employed to solve the underlying partial differential equation is unaffected by the cell shape or the presence of multiple shapes in the same mesh, greater freedom in mesh construction is allowed. In this case, an arbitrary polyhedral mesh can be used in which each cell can have any number of points and faces. Figure 1.4 shows part of the interior of a three dimensional domain meshed with tetrahedra, hexahedra, and arbitrary polyhedra.

The field of computational fluid dynamics (CFD) where the finite volume method is quite prevalent, has recently made great strides in utilizing arbitrary polyhedral meshes. Fluent, Star-CCM+, and OpenFOAM, three of the leading code packages for CFD, all provide utilities for constructing meshes of this type. Recent studies have shown increased accuracy and convergence rates for arbitrary polyhedral meshes in

CFD.[10][11][12] This provides incentive for their use in particle transport solutions as well, since coupling particle transport to fluid dynamics in multi-physics codes is of great interest, and using the same mesh for each set of physics would be preferable.



(a) Tetrahedral (b) Hexahedral (c) Arbitrary polyhedral

Figure 1.4: Slice through the center plane of various meshes of a simple two-material geometry.

Beyond the advantages that arbitrary polyhedral meshes may have in CFD, there is potentially an even greater advantage for particle transport due to the seven dimensional phase space of the angular flux. The enormity of the angular flux solution in computer memory is one reason why high fidelity particle transport calculations cannot be performed on a typical desktop computer, and it is also a motivation to pursue the highest possible ratio of accuracy to the number of unknowns. If the solution can be stored on a per cell basis, but the accuracy of the method is determined by the number of faces in the mesh, the accuracy to unknown ratio can be improved substantially. In face based methods, such as the SBA and its extensions proposed here, this is indeed the case; the solution is stored on a per cell basis, while the accuracy of the calculation in a given cell increases as the number of faces in the cell increases. Considering that polyhedral meshes will have roughly five times less cells than a tetrahedral mesh for the same number of faces if the polyhedral mesh is the Voronoi dual of the tetrahedral mesh, this means we could potentially get roughly the same accuracy while storing five times fewer unknowns.

1.3 Dissertation Layout

The purpose of this chapter has been to introduce the reader to the transport equation, motivate the need for its numerical solution, and briefly discuss how such solutions are obtained. In addition, capabilities and properties that future methods and implementations should exhibit have been discussed, and two of these capabilities have been singled out for further examination in the research presented here. Subsequent chapters are organized as follows.

Chapter 2 will be a review of some of the most common deterministic methods for solving the transport equation. This includes discretization schemes for the seven dimensional angular flux phase space. It will also include a discussion of the SBA, which this research builds upon. The SBA was chosen as a starting point for this research because of its ability to handle arbitrary polyhedral meshes, and because it adds additional concurrency to traditional balance methods, leading to more opportunities for parallelization and perhaps acceleration via GPGPU programming.

Chapter 3 will discuss changes that this research applies to the traditional SBA. This includes the addition of sub-slices in order to more accurately treat the streaming term of the transport equation and the application of the linear discontinuous finite element method on a per slice basis. At the end of this chapter, the capability to handle arbitrary polyhedral meshes will have been treated, while improving the accuracy of the traditional SBA given any underlying spatial discretization scheme, with relatively little added computation required.

Chapter 4 focuses on the capability to run efficiently on the next generation of super-computers. While Chapter 3 focuses mainly on theory and derivations, Chapter 4 is concerned with implementation. It will discuss two strategies to parallelize the solution over many nodes, each with multiple cores and perhaps even a GPU. These strategies can only be achieved through the framework of the extended SBA which is the result of this research, and the implications of these strategies will be discussed.

Chapter 5 will present results generated by the Slice-T code, which has been developed as a result of this research. These will include comparisons of the extended SBA to the traditional SBA in both accuracy and scalability, for the linear discontinuous finite element and diamond difference spatial discretization schemes. These results will then be discussed in Chapter 6 where final conclusions will be drawn and future work will be proposed.

2. REVIEW OF DETERMINISTIC SOLUTION METHODS

The purpose of this chapter is to present some of the most common discretization schemes and iterative methods for deterministic transport solutions. This includes common discretization schemes for the seven-dimensional angular flux phase space, in which time, energy, angle, and space are each discretized independently. It also includes iterative techniques to account for particle scattering, and methods for accelerating these iterative techniques when they are slow to converge. Finally, the parallel transport sweep is introduced as a scalable way to parallelize the solution on super-computers comprising hundreds of thousands, or even millions of cores. The transport equation that these methods aim to solve is repeated below for convenience.

$$\left(\frac{1}{v(E)}\frac{\partial}{\partial t} + \mathbf{\Omega}\cdot\nabla + \sigma_t\left(\mathbf{r},E,t\right)\right)\psi\left(\mathbf{r},\mathbf{\Omega},E,t\right) = \iint_{4\pi}\int_0^\infty \sigma_s\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E,t\right)\psi\left(\mathbf{r},\mathbf{\Omega}',E',t\right)dE'd\Omega' + q\left(\mathbf{r},\mathbf{\Omega},E,t\right) \quad (2.1)$$

2.1 Time Discretization

The temporal domain is typically discretized by solving for the angular flux solution at discrete points in time t_n, where $n = 0, 1, \ldots, N$. The discretization scheme is obtained by first integrating equation 2.1 from t_n to t_{n+1}, and dividing by $\Delta t_n = t_{n+1} - t_n$, which is equivalent to time-averaging the transport equation over the n-th time step. Using the notation $\psi_n\left(\mathbf{r},\mathbf{\Omega},E\right) = \psi\left(\mathbf{r},\mathbf{\Omega},E,t_n\right)$, the result of this time averaging is

$$\frac{\psi_{n+1}\left(\mathbf{r},\mathbf{\Omega},E\right)-\psi_{n}\left(\mathbf{r},\mathbf{\Omega},E\right)}{v(E)\Delta t_{n}}+\mathbf{\Omega}\cdot\nabla\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)+\overline{\sigma}_{t}\left(\mathbf{r},E\right)\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)=$$

$$\iint_{4\pi}\int_{0}^{\infty}\overline{\sigma}_{s}\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E\right)\overline{\psi}\left(\mathbf{r},\mathbf{\Omega}',E'\right)dE'd\Omega'+\overline{q}\left(\mathbf{r},\mathbf{\Omega},E\right)\ , \quad (2.2)$$

where

$$\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)=\frac{1}{\Delta t_{n}}\int_{t_{n}}^{t_{n+1}}\psi\left(\mathbf{r},\mathbf{\Omega},E,t\right)dt\ ,$$

$$\overline{q}\left(\mathbf{r},\mathbf{\Omega},E\right)=\frac{1}{\Delta t_{n}}\int_{t_{n}}^{t_{n+1}}q\left(\mathbf{r},\mathbf{\Omega},E,t\right)dt\ ,$$

$$\overline{\sigma}_{t}\left(\mathbf{r},E\right)=\frac{\displaystyle\int_{t_{n}}^{t_{n+1}}\sigma_{t}\left(\mathbf{r},E,t\right)\psi\left(\mathbf{r},\mathbf{\Omega},E,t\right)dt}{\displaystyle\int_{t_{n}}^{t_{n+1}}\psi\left(\mathbf{r},\mathbf{\Omega},E,t\right)dt}\ ,$$

and

$$\overline{\sigma}_{s}\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E\right)=\frac{\displaystyle\int_{t_{n}}^{t_{n+1}}\sigma_{s}\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E,t\right)\psi\left(\mathbf{r},\mathbf{\Omega},E,t\right)dt}{\displaystyle\int_{t_{n}}^{t_{n+1}}\psi\left(\mathbf{r},\mathbf{\Omega},E,t\right)dt}\ .$$

The first approximation often made at this point is to assume that the total and scattering cross sections have no time dependence over the time step, while still allowing their values to change between time steps. This allows them to be pulled out of the integrals in the definitions above. This is often a valid assumption, especially if the time steps are small enough that the temperature and composition of the background material does not have enough time to change significantly over the time step duration. Even when this assumption is not valid, iterative methods can be used wherein the cross section time dependence is ignored within each iteration. With this assumption, we can write the time averaged transport equation as

$$\frac{\psi_{n+1}(\mathbf{r},\mathbf{\Omega},E)-\psi_n(\mathbf{r},\mathbf{\Omega},E)}{v(E)\Delta t_n}+\mathbf{\Omega}\cdot\nabla\overline{\psi}(\mathbf{r},\mathbf{\Omega},E)+\sigma_{t,n}(\mathbf{r},E)\,\overline{\psi}(\mathbf{r},\mathbf{\Omega},E)=$$
$$\iint_{4\pi}\int_0^\infty \sigma_{s,n}(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E)\,\overline{\psi}(\mathbf{r},\mathbf{\Omega}',E')\,dE'd\Omega'+\overline{q}(\mathbf{r},\mathbf{\Omega},E)\ , \quad (2.3)$$

where $\sigma_{t,n}(\mathbf{r},E)$ and $\sigma_{s,n}(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E)$ are the total interaction and scattering cross sections respectively during time step n.

The next approximation is to represent the time averaged angular flux as a weighted average of the angular flux at the beginning and end of the time step. This can be written as

$$\overline{\psi}(\mathbf{r},\mathbf{\Omega},E)=\beta\,\psi_{n+1}(\mathbf{r},\mathbf{\Omega},E)+(1-\beta)\,\psi_n(\mathbf{r},\mathbf{\Omega},E)\ , \quad (2.4)$$

where β is a weighting parameter that chooses the discretization scheme. A value of $\beta=1$ results in the implicit Euler scheme, a value of $\beta=0$ results in the explicit Euler scheme, and a value of $\beta=1/2$ results in the Crank-Nicolson scheme. Both Euler schemes are first order accurate in time, meaning that the error in the solution is proportional to Δt, while the Crank-Nicolson scheme is second order accurate with error proportional to $(\Delta t)^2$. Both the implicit Euler and Crank-Nicolson schemes are unconditionally stable, while the explicit Euler scheme is likely to diverge for longer time steps.[13]

Rearranging equation 2.4 to solve for $\psi_{n+1}(\mathbf{r},\mathbf{\Omega},E)$ gives

$$\psi_{n+1}(\mathbf{r},\mathbf{\Omega},E)=\frac{1}{\beta}\overline{\psi}(\mathbf{r},\mathbf{\Omega},E)+\frac{(\beta-1)}{\beta}\psi_n(\mathbf{r},\mathbf{\Omega},E)\ , \quad (2.5)$$

and plugging this into equation 2.3 gives

$$\frac{\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)-\psi_n\left(\mathbf{r},\mathbf{\Omega},E\right)}{v(E)\Delta t_n\beta}+\mathbf{\Omega}\cdot\nabla\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)+\sigma_{t,n}\left(\mathbf{r},E\right)\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)=$$

$$\iint_{4\pi}\int_0^\infty \sigma_{s,n}\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E\right)\overline{\psi}\left(\mathbf{r},\mathbf{\Omega}',E'\right)dE'd\Omega'+\overline{q}\left(\mathbf{r},\mathbf{\Omega},E\right)\ . \quad (2.6)$$

If we further define

$$\sigma_{t,n,\text{eff}}\left(\mathbf{r},E\right)=\sigma_{t,n}\left(\mathbf{r},E\right)+\frac{1}{v(E)\Delta t_n\beta}$$

and

$$\overline{q}_{\text{eff}}\left(\mathbf{r},\mathbf{\Omega},E\right)=\overline{q}\left(\mathbf{r},\mathbf{\Omega},E\right)+\frac{1}{v(E)\Delta t_n\beta}\psi_n\left(\mathbf{r},\mathbf{\Omega},E\right)\ ,$$

we can rewrite equation 2.6 as

$$\mathbf{\Omega}\cdot\nabla\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)+\sigma_{t,n,\text{eff}}\left(\mathbf{r},E\right)\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)=$$

$$\iint_{4\pi}\int_0^\infty \sigma_{s,n}\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E\right)\overline{\psi}\left(\mathbf{r},\mathbf{\Omega}',E'\right)dE'd\Omega'+\overline{q}_{\text{eff}}\left(\mathbf{r},\mathbf{\Omega},E\right)\ . \quad (2.7)$$

Suppose for a moment that we were to make the following notational changes for convenience:

$$\overline{\psi}\left(\mathbf{r},\mathbf{\Omega},E\right)\to\psi\left(\mathbf{r},\mathbf{\Omega},E\right)\ ,$$

$$\overline{q}_{\text{eff}}\left(\mathbf{r},\mathbf{\Omega},E\right)\to q\left(\mathbf{r},\mathbf{\Omega},E\right)\ ,$$

$$\sigma_{t,n,\text{eff}}\left(\mathbf{r},E\right)\to\sigma_t\left(\mathbf{r},E\right)\ ,$$

and

$$\sigma_{s,n}\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E\right)\to\sigma_s\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E\right)\ .$$

We would then be able to rewrite equation 2.7 as

$$\boldsymbol{\Omega} \cdot \nabla \psi \left(\mathbf{r}, \boldsymbol{\Omega}, E\right) + \sigma_t \left(\mathbf{r}, E\right) \psi \left(\mathbf{r}, \boldsymbol{\Omega}, E\right) =$$

$$\iint_{4\pi} \int_0^\infty \sigma_s \left(\mathbf{r}, \boldsymbol{\Omega}' \cdot \boldsymbol{\Omega}, E' \to E\right) \psi \left(\mathbf{r}, \boldsymbol{\Omega}', E'\right) dE' d\Omega' + q \left(\mathbf{r}, \boldsymbol{\Omega}, E\right) \ . \quad (2.8)$$

This is an important result because equation 2.8 is the steady state version of equation 2.1 where the angular flux, particle source, and all cross sections are time-independent. Thus, after discretizing the temporal domain in the time dependent transport equation, we end up with a series of steady state problems to solve, one for each time step. The solution proceeds in the following steps:

1. Evaluate $\sigma_{t,n,\text{eff}} \left(\mathbf{r}, E\right)$ and $\overline{q}_{\text{eff}} \left(\mathbf{r}, \boldsymbol{\Omega}, E\right)$ using the angular flux from the previous time step, or the initial condition at the start of the problem.
2. Solve the resulting steady state equation (equation 2.7), for $\overline{\psi} \left(\mathbf{r}, \boldsymbol{\Omega}, E\right)$.
3. Use equation 2.5 and $\overline{\psi} \left(\mathbf{r}, \boldsymbol{\Omega}, E\right)$ to calculate $\psi_{n+1} \left(\mathbf{r}, \boldsymbol{\Omega}, E\right)$.
4. Return to step 1 for the next time step until the end of the temporal domain is reached.

2.2 Energy Discretization

With the important result at the end of the last section, that time dependent problems require only repeated solutions of steady state problems, energy discretization via the multi-group method will be demonstrated using the steady state transport equation which was given at the end of the last section as equation 2.8.

The first step is to divide the energy domain into G non-overlapping intervals called groups, with the upper bound of the highest energy group being some maximum energy E_0, above which it is assumed that there are no particles. Similarly, the lower bound of the lowest group is some minimum energy E_G, below which it is assumed that there are no particles. Note that the group numbering is in some sense reversed, as the highest energy is E_0 and the highest energy group is labeled group 1, while the lowest energy is E_G, and the lowest energy group is labeled group G. This non-intuitive convention is due to the fact that particles are typically born at high energies, and slow down to lower energies via scattering interactions occurring over their lifetime.

The discretization proceeds by integrating equation 2.8 over the g-th energy group spanning from lower bound E_g to upper bound E_{g-1}. The result of this integration is

$$\begin{aligned}\mathbf{\Omega}\cdot\nabla\int_{E_g}^{E_{g-1}}\psi\left(\mathbf{r},\mathbf{\Omega},E\right)dE+\int_{E_g}^{E_{g-1}}\sigma_t\left(\mathbf{r},E\right)\psi\left(\mathbf{r},\mathbf{\Omega},E\right)dE=\\ \iint_{4\pi}\int_0^{\infty}\psi\left(\mathbf{r},\mathbf{\Omega}',E'\right)\int_{E_g}^{E_{g-1}}\sigma_s\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E\right)dE\,dE'\,d\Omega'+\\ \int_{E_g}^{E_{g-1}}q\left(\mathbf{r},\mathbf{\Omega},E\right)dE\ .\end{aligned} \tag{2.9}$$

We then make the following definitions:

$$\psi_g\left(\mathbf{r},\mathbf{\Omega}\right)=\int_{E_g}^{E_{g-1}}\psi\left(\mathbf{r},\mathbf{\Omega},E\right)dE\ ,$$

$$q_g\left(\mathbf{r},\mathbf{\Omega}\right)=\int_{E_g}^{E_{g-1}}q\left(\mathbf{r},\mathbf{\Omega},E\right)dE\ ,$$

and

$$\sigma_s\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to g\right)=\int_{E_g}^{E_{g-1}}\sigma_s\left(\mathbf{r},\mathbf{\Omega}'\cdot\mathbf{\Omega},E'\to E\right)dE\ ,$$

in order to rewrite equation 2.9 as

$$\mathbf{\Omega} \cdot \nabla \psi_g(\mathbf{r}, \mathbf{\Omega}) + \int_{E_g}^{E_{g-1}} \sigma_t(\mathbf{r}, E)\, \psi(\mathbf{r}, \mathbf{\Omega}, E)\, dE =$$

$$\sum_{g'=1}^{G} \iint_{4\pi} \int_{E_{g'}}^{E_{g'-1}} \sigma_s(\mathbf{r}, \mathbf{\Omega}' \cdot \mathbf{\Omega}, E' \to g)\, \psi(\mathbf{r}, \mathbf{\Omega}', E')\, dE' d\Omega' + q_g(\mathbf{r}, \mathbf{\Omega}) \ , \quad (2.10)$$

where the integral over all energies in the scattering term has been replaced by a sum of integrals over each group in the energy domain. The next step is to define

$$\sigma_{t,g}(\mathbf{r}, \mathbf{\Omega}) = \frac{\displaystyle\int_{E_g}^{E_{g-1}} \sigma_t(\mathbf{r}, E)\, \psi(\mathbf{r}, \mathbf{\Omega}, E)\, dE}{\displaystyle\int_{E_g}^{E_{g-1}} \psi(\mathbf{r}, \mathbf{\Omega}, E)\, dE}$$

and

$$\sigma_{s,g' \to g}(\mathbf{r}, \mathbf{\Omega}' \cdot \mathbf{\Omega}) = \frac{\displaystyle\int_{E_{g'}}^{E_{g'-1}} \sigma_s(\mathbf{r}, \mathbf{\Omega}' \cdot \mathbf{\Omega}, E' \to g)\, \psi(\mathbf{r}, \mathbf{\Omega}', E')\, dE'}{\displaystyle\int_{E_{g'}}^{E_{g'-1}} \psi(\mathbf{r}, \mathbf{\Omega}', E')\, dE'} \ ,$$

in order to rewrite equation 2.10 as

$$\mathbf{\Omega} \cdot \nabla \psi_g(\mathbf{r}, \mathbf{\Omega}) + \sigma_{t,g}(\mathbf{r}, \mathbf{\Omega})\, \psi_g(\mathbf{r}, \mathbf{\Omega}) =$$

$$\sum_{g'=1}^{G} \iint_{4\pi} \sigma_{s,g' \to g}(\mathbf{r}, \mathbf{\Omega}' \cdot \mathbf{\Omega})\, \psi_{g'}(\mathbf{r}, \mathbf{\Omega}')\, d\Omega' + q_g(\mathbf{r}, \mathbf{\Omega}) \ , \quad (2.11)$$

It is important to note that up to this point, no approximations have been made. We have only integrated the steady state transport equation over group g and made some convenient definitions. Unfortunately, even though the total interaction cross section $\sigma_t(\mathbf{r}, E)$ was not dependent on angle, the weighted cross section $\sigma_{t,g}(\mathbf{r}, \mathbf{\Omega})$

is angle dependent because it has been weighted by the angular flux which is angle dependent. In addition, the weighted cross sections are defined using the angular flux, which is the function we are trying to solve for. The approximation that makes the multi-group method feasible is to assume first that the angular flux is separable in energy within each group, which can be written as

$$\psi\left(\mathbf{r},\boldsymbol{\Omega},E\right)=\Psi\left(\mathbf{r},\boldsymbol{\Omega}\right)f_g(E) \quad \text{for} \quad E_g<E<E_{g-1}\ , \tag{2.12}$$

and that the energy shape functions $f_g(E)$ can be guessed with reasonable accuracy to calculate the cross sections in each group. If these approximations are made, the multi-group cross section definitions can be rewritten as

$$\sigma_{t,g}\left(\mathbf{r}\right)=\frac{\displaystyle\int_{E_g}^{E_{g-1}}\sigma_t\left(\mathbf{r},E\right)f_g\left(E\right)dE}{\displaystyle\int_{E_g}^{E_{g-1}}f_g\left(E\right)dE}$$

and

$$\sigma_{s,g'\to g}\left(\mathbf{r},\boldsymbol{\Omega}'\cdot\boldsymbol{\Omega}\right)=\frac{\displaystyle\int_{E_{g'}}^{E_{g'-1}}\sigma_s\left(\mathbf{r},\boldsymbol{\Omega}'\cdot\boldsymbol{\Omega},E'\to g\right)f_g\left(E'\right)dE'}{\displaystyle\int_{E_{g'}}^{E_{g'-1}}f_g\left(E'\right)dE'}\ .$$

Using these definitions, we can now write down the steady state multi-group transport equations as

$$\boldsymbol{\Omega}\cdot\nabla\,\psi_g\left(\mathbf{r},\boldsymbol{\Omega}\right)+\sigma_{t,g}\left(\mathbf{r}\right)\psi_g\left(\mathbf{r},\boldsymbol{\Omega}\right)=$$

$$\sum_{g'=1}^{G}\iint_{4\pi}\sigma_{s,g'\to g}\left(\mathbf{r},\boldsymbol{\Omega}'\cdot\boldsymbol{\Omega}\right)\psi_{g'}\left(\mathbf{r},\boldsymbol{\Omega}'\right)d\Omega'+q_g\left(\mathbf{r},\boldsymbol{\Omega}\right)\ , \tag{2.13}$$

where $g = 1, 2, \ldots, G$. Performing this energy integration for each group results in a set of G integro-partial differential equations, which are coupled by the scattering term on the right. A common method for solving the multi-group equations is the Gauss-Seidel iterative approach whereby each equation is solved, starting with group 1, using the most recent estimate of $\psi_g(\mathbf{r}, \mathbf{\Omega})$ in the other groups. It is easy to see that if there is no up-scattering, i.e. $\sigma_{s,g' \to g}(\mathbf{r}, \mathbf{\Omega}' \cdot \mathbf{\Omega}) = 0$ for $g' > g$, this iterative scheme will converge in a single iteration.

The number of energy groups typically used in deterministic neutron transport is on the order of 10^2. The International Reactor Dosimetry and Fusion File (IRDFF) group structure has only 640 energy groups, and this is considered to be on the high end of energy resolution. Even with this many groups, the complex energy dependence of neutron interaction cross sections, such as the total interaction cross section for ^{238}U, is difficult to represent accurately as shown in Figure 2.1. This is primarily due to the presence of resonances in the neutron cross sections, where the cross section varies wildly over small ranges in energy. In addition, the assumption that the angular flux was separable in energy within each group becomes less valid as the group widths get larger. It is also very difficult to quantify the error introduced by the multi-group method since it does not show convergence with refinement until the group count is impractically large, whereas convergence is observed with practical spatial and temporal discretizations.

Of course, there is also the issue of where to get the shape functions $f_g(E)$. Typically, some information about the problem will lead to an initial guess at the shape function. For instance, in a nuclear power reactor, the energy spectrum of neutrons can be crudely represented by the combination of a Maxwellian spectrum at low energies, a $1/E$ spectrum at intermediate energies, and a Watt fission spectrum at high energies. Multi-group cross section generation codes such as NJOY provide such

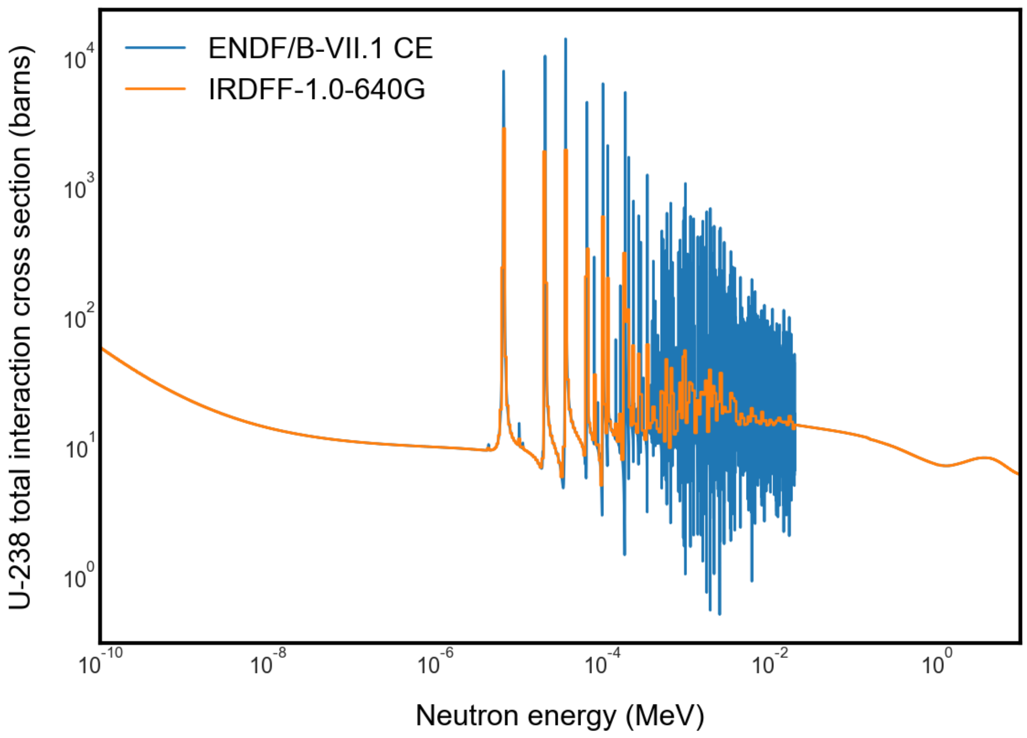


Figure 2.1: Total interaction cross section for ^{238}U.

spectra, and their relative magnitudes can be specified. Iterations can be performed where more detailed spectra can be obtained from the solution, and used to refine the postulated $f_g(E)$ to produce more accurate multi-group cross sections.

2.3 Angular Discretization

Before discussing the discrete ordinates (S_N) and spherical harmonics (P_N) angular discretization schemes, we must first address the scattering term on the right hand side of the multi-group equations. Any angular discretization scheme will inevitably restrict the directions in which particles can travel either explicitly or by limiting the complexity of the solution's angular dependence. While $\mathbf{\Omega}' \cdot \mathbf{\Omega}$ could be computed for each pair of discrete directions, and the exact value of the scattering cross section could be evaluated for each pair, it is more convenient for this discussion to approximate the angular dependence of the scattering cross sections. We begin by defining $\mu = \mathbf{\Omega}' \cdot \mathbf{\Omega}$ and the scattering operator $\mathbf{S}$ as

$$\mathbf{S}\psi_{g'}\left(\mathbf{r}, \mathbf{\Omega}'\right) = \iint_{4\pi} \sigma_{s,g' \to g}\left(\mathbf{r}, \mu\right) \psi_{g'}\left(\mathbf{r}, \mathbf{\Omega}'\right) d\Omega' \ ,$$

which appears inside the energy group summation in the scattering term of equation 2.13. Throughout this section we will be making use of the real spherical harmonic functions, also known as the tesseral spherical harmonics, which are defined here as

$$Y_j^k\left(\mathbf{\Omega}\right) = \begin{cases} \sqrt{c_j^k} P_j^k(\xi) \cos(k\omega), & 0 \le k \le j \\ \sqrt{c_j^k} P_j^{|k|}(\xi) \sin(|k|\omega), & -j \le k < 0 \ , \end{cases}$$

where ξ is the polar angle associated with $\mathbf{\Omega}$, ω is the azimuthal angle associated with $\mathbf{\Omega}$, $P_j^k(\xi)$ are the associated Legendre polynomials, and

$$c_j^k = (2 - \delta_{k,0}) \frac{(j - |k|)!}{(j + |k|)!} \ .$$

In order to derive a useful result, we apply the scattering operator to the spherical harmonic function $Y_j^k\left(\mathbf{\Omega}'\right)$

$$\mathbf{S}Y_j^k\left(\mathbf{\Omega}'\right) = \iint_{4\pi} \sigma_{s,g' \to g}\left(\mathbf{r}, \mu\right) Y_j^k\left(\mathbf{\Omega}'\right) d\Omega' \ . \tag{2.14}$$

We then perform a Legendre polynomial expansion of the scattering cross section to arrive at

$$\mathbf{S}Y_j^k\left(\mathbf{\Omega}'\right) = \iint_{4\pi} \sum_{l=0}^{\infty} \frac{2l+1}{4\pi} \sigma_{l,s,g' \to g}\left(\mathbf{r}\right) P_l^0(\mu) Y_j^k\left(\mathbf{\Omega}'\right) d\Omega' \ , \tag{2.15}$$

where

$$\sigma_{l,s,g' \to g}\left(\mathbf{r}\right) = 2\pi \int_{-1}^{1} \sigma_{s,g' \to g}\left(\mathbf{r}, \mu\right) P_l^0(\mu) d\mu \ . \tag{2.16}$$

We can now use the addition theorem for spherical harmonics to replace the Legendre polynomials by a sum of products of spherical harmonic functions

$$P_l^0(\mu) = P_l^0\left(\mathbf{\Omega}' \cdot \mathbf{\Omega}\right) = \sum_{m=-l}^{+l} Y_l^m\left(\mathbf{\Omega}'\right) Y_l^m\left(\mathbf{\Omega}\right) \ , \tag{2.17}$$

and plug this into equation 2.15 to arrive at

$$\mathbf{S}Y_k^j\left(\mathbf{\Omega}'\right) = \sum_{l=0}^{\infty} \frac{2l+1}{4\pi} \sigma_{l,s,g'\to g}\left(\mathbf{r}\right) \sum_{m=-l}^{+l} Y_l^m\left(\mathbf{\Omega}\right) \iint_{4\pi} Y_l^m\left(\mathbf{\Omega}'\right) Y_j^k\left(\mathbf{\Omega}'\right) d\Omega' \ . \tag{2.18}$$

Using the orthogonality of the spherical harmonics

$$\iint_{4\pi} Y_l^m\left(\mathbf{\Omega}\right) Y_j^k\left(\mathbf{\Omega}\right) d\Omega = \frac{4\pi}{2l+1} \delta_{l,j}\delta_{m,k} \ , \tag{2.19}$$

only a single term survives the infinite sum, giving the useful result we were seeking

$$\mathbf{S}Y_j^k\left(\mathbf{\Omega}'\right) = \sigma_{j,s,g'\to g}\left(\mathbf{r}\right) Y_j^k\left(\mathbf{\Omega}\right) \ . \tag{2.20}$$

To use this result, we first expand $\psi_{g'}\left(\mathbf{r}, \mathbf{\Omega}'\right)$ in the spherical harmonic functions

$$\psi_{g'}\left(\mathbf{r}, \mathbf{\Omega}'\right) = \sum_{l=0}^{\infty} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \phi_{l,g'}^m\left(\mathbf{r}\right) Y_l^m\left(\mathbf{\Omega}'\right) \ , \tag{2.21}$$

where

$$\phi_{l,g'}^m\left(\mathbf{r}\right) = \iint_{4\pi} Y_l^m\left(\mathbf{\Omega}\right) \psi_{g'}\left(\mathbf{r}, \mathbf{\Omega}\right) d\Omega \ . \tag{2.22}$$

We can now apply the scattering operator to both sides of equation 2.21 and use the result in equation 2.20 to write

$$\mathbf{S}\psi_{g'}\left(\mathbf{r},\mathbf{\Omega}'\right) = \sum_{l=0}^{\infty}\sum_{m=-l}^{+l}\frac{2l+1}{4\pi}\sigma_{l,s,g'\to g}\left(\mathbf{r}\right)\phi_{l,g'}^{m}\left(\mathbf{r}\right)Y_{l}^{m}\left(\mathbf{\Omega}\right)\ . \tag{2.23}$$

Plugging this result into equation 2.13 gives

$$\mathbf{\Omega}\cdot\nabla\,\psi_g\left(\mathbf{r},\mathbf{\Omega}\right)+\sigma_{t,g}\left(\mathbf{r}\right)\psi_g\left(\mathbf{r},\mathbf{\Omega}\right) = \sum_{g'=1}^{G}\sum_{l=0}^{\infty}\sum_{m=-l}^{+l}\frac{2l+1}{4\pi}\sigma_{l,s,g'\to g}\left(\mathbf{r}\right)\phi_{l,g'}^{m}\left(\mathbf{r}\right)Y_{l}^{m}\left(\mathbf{\Omega}\right)+\sum_{l=0}^{\infty}\sum_{m=-l}^{+l}\frac{2l+1}{4\pi}q_{l,g}^{m}\left(\mathbf{r}\right)Y_{l}^{m}\left(\mathbf{\Omega}\right)\ . \tag{2.24}$$

where we have also expanded the external source rate density in the spherical harmonic functions with expansion coefficients given by

$$q_{l,g}^{m}\left(\mathbf{r}\right) = \iint_{4\pi}Y_{l}^{m}\left(\mathbf{\Omega}\right)q_g\left(\mathbf{r},\mathbf{\Omega}\right)d\Omega\ . \tag{2.25}$$

It is again important to note that thus far in this section, no approximations have been made. We have simply used the properties of the Legendre polynomials and spherical harmonics to obtain multi-group transport equations that no longer depend directly on $\mathbf{\Omega}'$ in the scattering term. Of course, the integral in the scattering term has now become an infinite sum, which is not exactly ideal either. The approximation that must be made in order to use equation 2.24 is to truncate the infinite sums at specified values L_1 and L_2

$$\mathbf{\Omega} \cdot \nabla \psi_g(\mathbf{r}, \mathbf{\Omega}) + \sigma_{t,g}(\mathbf{r}) \psi_g(\mathbf{r}, \mathbf{\Omega}) = \sum_{g'=1}^{G} \sum_{l=0}^{L_1} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s,g' \to g}(\mathbf{r}) \phi_{l,g'}^m(\mathbf{r}) Y_l^m(\mathbf{\Omega}) + \sum_{l=0}^{L_2} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} q_{l,g}^m(\mathbf{r}) Y_l^m(\mathbf{\Omega}) \ . \quad (2.26)$$

It should be noted that there are two scenarios that make this approximation valid for the scattering term. The first is if the angular flux of the true solution is able to be accurately represented by a spherical harmonic expansion up to degree L_1. The second is if the scattering cross section is able to be accurately represented by a Legendre polynomial expansion up to degree L_1. In each case, either $\phi_{l,g'}^m(\mathbf{r}) \approx 0$ for $l > L_1$, or $\sigma_{l,s,g' \to g}(\mathbf{r}) \approx 0$ for $l > L_1$ respectively, causing the true infinite summation in the scattering term to effectively stop at degree L_1. In addition, many sources are likely to be isotropic, in which case the last summation in equation 2.26 can be accurately represented by a single term, i.e. $L_2 = 0$. Equation 2.26 is the form of the multi-group transport equations that will be our starting point for discussing the S_N and P_N angular discretization schemes.

2.3.1 Discrete Ordinates

The simplest description of the S_N method is that the multi-group transport equations given in equation 2.26 are solved for a particular set of angles, and all angle integrated quantities are then estimated by quadrature summation. The transport equation for group g then becomes a set of N equations, resulting in a total of $N \times G$ equations, where N is the number of angles in the set. Using the notation $\psi_{g,n}(\mathbf{r}) = \psi_g(\mathbf{r}, \mathbf{\Omega}_n)$, this can be written as

$$\boldsymbol{\Omega}_n \cdot \nabla \psi_{g,n}(\mathbf{r}) + \sigma_{t,g}(\mathbf{r})\,\psi_{g,n}(\mathbf{r}) =$$

$$\sum_{g'=1}^{G}\sum_{l=0}^{L_1}\sum_{m=-l}^{+l}\frac{2l+1}{4\pi}\sigma_{l,s,g'\to g}(\mathbf{r})\,\phi_{l,g'}^{m}(\mathbf{r})\,Y_l^m(\boldsymbol{\Omega}_n) +$$

$$\sum_{l=0}^{L_2}\sum_{m=-l}^{+l}\frac{2l+1}{4\pi}q_{l,g}^{m}(\mathbf{r})\,Y_l^m(\boldsymbol{\Omega}_n)\ , \quad (2.27)$$

where $n = 1, 2, \ldots, N$. The accuracy of the S_N method is then largely determined by the quadrature rule chosen to select the angles in the set. The quadrature rule is a set of nodes and weights $\{\boldsymbol{\Omega}_n, \omega_n\}_{n=1}^{N}$, such that

$$\iint_{4\pi} f(\boldsymbol{\Omega})\,d\Omega \approx \sum_{n=1}^{N}\omega_n f(\boldsymbol{\Omega}_n)\ . \quad (2.28)$$

For example, to compute the coefficients in the spherical harmonic expansion of the angular flux in the scattering term, the integral would be approximated as

$$\phi_{l,g'}^{m}(\mathbf{r}) = \iint_{4\pi} Y_l^m(\boldsymbol{\Omega})\,\psi_{g'}(\mathbf{r},\boldsymbol{\Omega})\,d\Omega \approx \sum_{n=1}^{N}\omega_n Y_l^m(\boldsymbol{\Omega}_n)\,\psi_{g',n}(\mathbf{r})\ . \quad (2.29)$$

While there is significant freedom in choosing the quadrature rule used, there are some desirable properties that such rules should have. Three constraints typically imposed are

$$\sum_{n=1}^{N}\omega_n\boldsymbol{\Omega}_n = \mathbf{0}\ ,$$

$$\sum_{n=1}^{N}\omega_n = 4\pi\ ,$$

and

$$\sum_{n=1}^{N}\omega_n\Omega_{n,x}^2 = \sum_{n=1}^{N}\omega_n\Omega_{n,y}^2 = \sum_{n=1}^{N}\omega_n\Omega_{n,z}^2 = \frac{1}{3}\sum_{n=1}^{N}\omega_n\ ,$$

where $\Omega_{n,x}$, $\Omega_{n,y}$, and $\Omega_{n,z}$ are the Cartesian components of $\mathbf{\Omega}_n$. The first of these is imposed in order to enable the use of reflecting boundary conditions, while all three are imposed to guarantee exact integration of the zeroth and first angular moments of the transport equation in the case of a linearly anisotropic angular flux.

As an example, consider the Gauss-Chebyshev product quadrature, where the polar and azimuthal angles of the nodes $\mathbf{\Omega}_n$ are selected independently. The polar angles are selected by using a Gauss-Legendre quadrature rule for the cosine of the polar angle, while the azimuthal angles are chosen to be equally weighted and equally spaced from 0 to 2π. Although not required, it is usually the case that each octant contains equal numbers of polar and azimuthal nodes as shown in Figure 2.2 which shows the S_4 and S_8 Gauss-Chebyshev product quadrature nodes. The subscript denotes the number of nodes in the polar angle from 0 to π. While many other quadrature sets exist, Gauss-Chebyshev quadrature sets are quite common, and are used exclusively in this research.

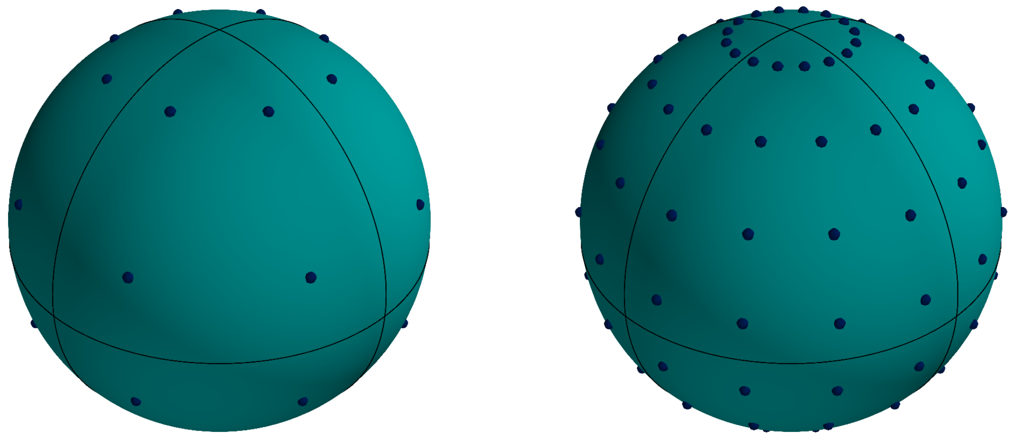
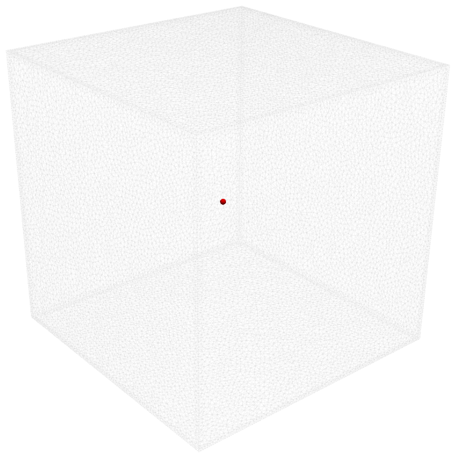

(a) S_4 (b) S_8

Figure 2.2: Gauss-Chebyshev quadrature nodes.

As discussed in Chapter 1, a significant drawback to the discrete ordinates method which solves the transport equation for a specific set of angles is that it inherently changes the physics represented by the equation. By discretizing in angle based on collocation as is done above, particles are restricted to travel only in the directions in the quadrature set. This leads to ray effects, which are most prominent when there is very little scattering and in the vicinity of localized sources. Consider for instance an isotropic point source in a vacuum. The scalar flux in this situation should exhibit spherical symmetry, however in a discrete ordinates calculation, the scalar flux would appear to have rays emanating from the source along the directions of the quadrature set. This is demonstrated qualitatively in Figure 2.3 which shows a small localized source of strength 10 p/cm^3·s and the contour plot of where the scalar flux has dropped to 0.04 p/cm^2·s. Analytically, this contour plot would be a perfect sphere.


(a) Source location

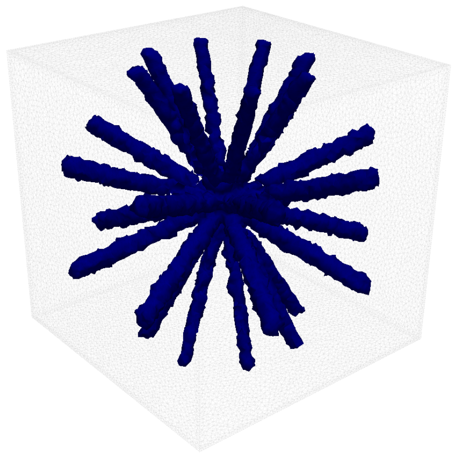

(b) Contour surface

Figure 2.3: Localized source of strength 10 p/cm^3·s placed at the center of a box of side length 200 cm and a contour surface showing where the scalar flux drops to 0.04 p/cm^2·s generated by Slice-T using an S_4 quadrature set.

2.3.2 Spherical Harmonics

The P_N method proceeds from equation 2.24 by expanding the angular flux on the left hand side in the spherical harmonics functions as was done for the scattering term using equations 2.21 and 2.22, resulting in

$$\left(\mathbf{\Omega} \cdot \nabla + \sigma_{t,g}(\mathbf{r})\right) \sum_{l=0}^{\infty} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \phi_{l,g}^{m}(\mathbf{r}) Y_l^m(\mathbf{\Omega}) =$$

$$\sum_{g'=1}^{G} \sum_{l=0}^{\infty} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s,g'\to g}(\mathbf{r}) \phi_{l,g'}^{m}(\mathbf{r}) Y_l^m(\mathbf{\Omega}) +$$

$$\sum_{l=0}^{\infty} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} q_{l,g}^{m}(\mathbf{r}) Y_l^m(\mathbf{\Omega}) \ . \quad (2.30)$$

We then multiply by $Y_j^k(\mathbf{\Omega})$, integrate over all angles, and take advantage of the orthogonality of the spherical harmonics to obtain

$$\sum_{l=0}^{\infty} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \iint_{4\pi} \mathbf{\Omega} \cdot \nabla \phi_{l,g}^{m}(\mathbf{r}) Y_j^k(\mathbf{\Omega}) Y_l^m(\mathbf{\Omega}) \, d\Omega +$$

$$\sigma_{t,g}(\mathbf{r}) \phi_{j,g}^{k}(\mathbf{r}) - \sum_{g'=1}^{G} \sigma_{j,s,g'\to g}(\mathbf{r}) \phi_{j,g'}^{k}(\mathbf{r}) = q_{j,g}^{k}(\mathbf{r}) \ . \quad (2.31)$$

To simplify this further, we note that

$$\mathbf{\Omega} = \left(\sqrt{1-\xi^2}\,\cos(\omega),\ \sqrt{1-\xi^2}\,\sin(\omega),\ \xi\right)^T$$

where again, ξ is the polar angle associated with $\mathbf{\Omega}$ and ω is the azimuthal angle associated with $\mathbf{\Omega}$. Using this definition, we can write

$$\boldsymbol{\Omega} \cdot \nabla \ = \sqrt{1-\xi^2}\ \cos(\omega)\frac{\partial}{\partial x} + \sqrt{1-\xi^2}\ \sin(\omega)\frac{\partial}{\partial y} + \xi\frac{\partial}{\partial z}\ , \tag{2.32}$$

and the last remaining integral in equation 2.31 can be expanded as

$$\begin{aligned}
&\iint_{4\pi} \boldsymbol{\Omega} \cdot \nabla\, \phi_{l,g}^{m}(\mathbf{r})\, Y_j^k(\boldsymbol{\Omega})\, Y_l^m(\boldsymbol{\Omega})\, d\Omega = \\
&\qquad \frac{\partial}{\partial x}\left(\phi_{l,g}^{m}(\mathbf{r}) \iint_{4\pi} \sqrt{1-\xi^2}\ \cos(\omega) Y_j^k(\boldsymbol{\Omega})\, Y_l^m(\boldsymbol{\Omega})\, d\Omega\right) + \\
&\qquad\qquad \frac{\partial}{\partial y}\left(\phi_{l,g}^{m}(\mathbf{r}) \iint_{4\pi} \sqrt{1-\xi^2}\ \sin(\omega) Y_j^k(\boldsymbol{\Omega})\, Y_l^m(\boldsymbol{\Omega})\, d\Omega\right) + \\
&\qquad\qquad\qquad \frac{\partial}{\partial z}\left(\phi_{l,g}^{m}(\mathbf{r}) \iint_{4\pi} \xi Y_j^k(\boldsymbol{\Omega})\, Y_l^m(\boldsymbol{\Omega})\, d\Omega\right)
\end{aligned} \tag{2.33}$$

This is useful because we can take advantage of the recursion relations shown on the next page with constants given by

$$A_j^k = \sqrt{\frac{(j-k+1)(j+k+1)}{(2j+3)(2j+1)}}\ , \qquad B_j^k = \sqrt{\frac{(j-k)(j+k)}{(2j+1)(2j-1)}}\ ,$$

$$C_j^k = \sqrt{\frac{(j+k+1)(j+k+2)}{(2j+3)(2j+1)}}\ , \qquad D_j^k = \sqrt{\frac{(j-k)(j-k-1)}{(2j+1)(2j-1)}}\ ,$$

$$E_j^k = \sqrt{\frac{(j-k+1)(j-k+2)}{(2j+3)(2j+1)}}\ , \qquad F_j^k = \sqrt{\frac{(j+k)(j+k-1)}{(2j+1)(2j-1)}}\ .$$

This allows each integral on the right hand side of equation 2.33 to become a sum of integrals whose integrands are products of spherical harmonics, which by orthogonality evaluate to constant values. The end result is that the streaming term in the transport equations has become a linear combination of coefficient spatial derivatives.

$$\xi Y_j^k(\mathbf{\Omega}) = A_j^k Y_{j+1}^k(\mathbf{\Omega}) + B_j^k Y_{j-1}^k(\mathbf{\Omega}) \tag{2.34}$$

$$\sqrt{1-\xi^2}\,\cos(\omega) Y_j^k(\mathbf{\Omega}) = \begin{cases} \frac{1}{\sqrt{2}}\left(C_j^0 Y_{j+1}^1(\mathbf{\Omega}) - D_j^0 Y_{j-1}^1(\mathbf{\Omega})\right), & k=0 \\ \frac{1}{2}\left(C_j^k Y_{j+1}^{k+1}(\mathbf{\Omega}) - D_j^k Y_{j-1}^{k+1}(\mathbf{\Omega})\right) + \begin{cases} \frac{1}{2}\left(-E_j^k Y_{j+1}^{k-1}(\mathbf{\Omega}) - F_j^k Y_{j-1}^{k-1}(\mathbf{\Omega})\right), & k>1 \\ \frac{1}{\sqrt{2}}\left(-E_j^k Y_{j+1}^{k-1}(\mathbf{\Omega}) - F_j^k Y_{j-1}^{k-1}(\mathbf{\Omega})\right), & k=1 \end{cases} \\ \frac{1}{2}\left(E_j^k Y_{j+1}^{k-1}(\mathbf{\Omega}) - F_j^k Y_{j-1}^{k-1}(\mathbf{\Omega})\right) + \begin{cases} \frac{1}{2}\left(-C_j^k Y_{j+1}^{k+1}(\mathbf{\Omega}) + D_j^k Y_{j-1}^{k+1}(\mathbf{\Omega})\right), & k<-1 \\ 0, & k=-1 \end{cases} \end{cases} \tag{2.35}$$

$$\sqrt{1-\xi^2}\,\sin(\omega) Y_j^k(\mathbf{\Omega}) = \begin{cases} \frac{1}{\sqrt{2}}\left(C_j^0 Y_{j+1}^{-1}(\mathbf{\Omega}) - D_j^0 Y_{j-1}^{-1}(\mathbf{\Omega})\right), & k=0 \\ \frac{1}{2}\left(C_j^k Y_{j+1}^{-k-1}(\mathbf{\Omega}) - D_j^k Y_{j-1}^{-k-1}(\mathbf{\Omega})\right) + \begin{cases} \frac{1}{2}\left(E_j^k Y_{j+1}^{-k+1}(\mathbf{\Omega}) - F_j^k Y_{j-1}^{-k+1}(\mathbf{\Omega})\right), & k>1 \\ 0, & k=1 \end{cases} \\ \frac{1}{2}\left(-E_j^k Y_{j+1}^{-k+1}(\mathbf{\Omega}) + F_j^k Y_{j-1}^{-k+1}(\mathbf{\Omega})\right) + \begin{cases} \frac{1}{2}\left(-C_j^k Y_{j+1}^{-k-1}(\mathbf{\Omega}) + D_j^k Y_{j-1}^{-k-1}(\mathbf{\Omega})\right), & k<-1 \\ \frac{1}{\sqrt{2}}\left(-C_j^k Y_{j+1}^{-k-1}(\mathbf{\Omega}) + D_j^k Y_{j-1}^{-k-1}(\mathbf{\Omega})\right), & k=-1 \end{cases} \end{cases} \tag{2.36}$$

This multiplication and integration is performed for every $Y_j^k(\mathbf{\Omega})$ resulting in an infinite set of equations for each energy group which can be written as

$$\left(\mathbf{A}_x \frac{\partial}{\partial x} + \mathbf{A}_y \frac{\partial}{\partial y} + \mathbf{A}_z \frac{\partial}{\partial z} + \sigma_{t,g}(\mathbf{r}) - \sum_{g'=1}^{G} \sigma_{j,s,g'\to g}(\mathbf{r})\right) \mathbf{\Phi}(\mathbf{r}) = \mathbf{Q}(\mathbf{r}) \ , \qquad (2.37)$$

where $\mathbf{A}_x$, $\mathbf{A}_y$, and $\mathbf{A}_z$ are matrices whose elements are linear combinations of A_j^k, B_j^k, C_j^k, D_j^k, E_j^k, and F_j^k, $\mathbf{\Phi}(\mathbf{r})$ is a vector containing the coefficient functions $\phi_{j,g}^k(\mathbf{r})$, and $\mathbf{Q}(\mathbf{r})$ is a vector containing the coefficient functions $q_{j,g}^k(\mathbf{r})$. Yet again, no approximations have been made in this section; we have simply used the orthogonality and recursion relations of the spherical harmonics to write the transport equation for group g as an infinite system of equations, which is not ideal either.

The approximation that must be made is the same as the one made when we modified the scattering term, namely to truncate the expansion at some level by setting $\phi_{N+1,g}^k(\mathbf{r}) = 0$, or expressing $\phi_{N+1,g}^k(\mathbf{r})$ in terms of lower-order moments. While the higher-order moments may not be zero, you don't need to know them in order to find the lower-order ones, and we end up with a finite system of equations for each energy group. To be exact, we end up with $(N+1)^2$ equations for each energy group. By the same logic as before, the accuracy of this approximation is related to how accurately the true angular flux can be represented by such a finite expansion. In addition, if the angular flux can be accurately represented by a truncated expansion where N is small, the P_N method is preferable over the S_N method. This will typically occur in optically thick problems with high scattering ratios, whereas problems that are dominated by the streaming term will be more accurately treated by the S_N method. The research presented here uses the S_N method for angular discretization, and the P_N method has only been presented in the interest of completeness and to provide a more thorough review of deterministic methods.

2.4 Spatial Discretization

For reasons that will become more clear later in this chapter when iterative methods are discussed, a discussion of spatial discretization methods will start from the fixed-source, steady state, energy independent transport equation

$$\mathbf{\Omega} \cdot \nabla \psi (\mathbf{r}, \mathbf{\Omega}) + \sigma_t (\mathbf{r}) \psi (\mathbf{r}, \mathbf{\Omega}) = q (\mathbf{r}, \mathbf{\Omega}) \ . \tag{2.38}$$

This can be justified by noting the following points

- Time dependent problems require only repeated solutions of steady-state problems as previously discussed.
- Iterative methods for the energy dependence result in iterations known as outer iterations, where the contribution to the scattering term for each group from all other groups is computed using information from a previous iteration. This effectively removes all energy dependence within each iteration.
- Iterative methods for the scattering term result in iterations known as inner iterations, where the contribution to the scattering term for each group from itself is computed using information from a previous iteration. This effectively converts the entire right hand side of the transport equation into a fixed source $q (\mathbf{r}, \mathbf{\Omega})$ within each iteration.

Furthermore, angle dependence can be discretized using the discrete ordinates method so that equation 2.38 can be written as

$$\mathbf{\Omega}_n \cdot \nabla \psi_n (\mathbf{r}) + \sigma_t (\mathbf{r}) \psi_n (\mathbf{r}) = q_n (\mathbf{r}) \ , \quad n = 1, 2, \ldots, N. \tag{2.39}$$

2.4.1 Finite Volume Methods

Finite volume methods aim to solve partial differential equations such as equation 2.39 by volume averaging the differential equation over each cell in the spatial mesh. Divergence or gradient terms are converted to surface integrals, and the flux through each surface is determined by an interpolation scheme using the cell averaged values for the cells on either side of the face. To illustrate this, we first integrate equation 2.39 over cell i in the mesh and divide by that cell's volume V_i, assuming that the total interaction cross section is constant within each cell of the mesh and denoted $\sigma_{t,i}$

$$\frac{1}{V_i}\,\mathbf{\Omega}_n \cdot \iiint_{V_i} \nabla\psi_n(\mathbf{r})\,d^3r + \sigma_{t,i}\frac{1}{V_i}\iiint_{V_i}\psi_n(\mathbf{r})\,d^3r = \frac{1}{V_i}\iiint_{V_i} q_n(\mathbf{r})\,d^3r\ . \tag{2.40}$$

The next step is to define

$$\psi_{n,i} = \frac{1}{V_i}\iiint_{V_i}\psi_n(\mathbf{r})\,d^3r$$

and

$$q_{n,i} = \frac{1}{V_i}\iiint_{V_i} q_n(\mathbf{r})\,d^3r\ .$$

We also convert the first volume integral into a surface integral to rewrite equation 2.40 as

$$\frac{1}{V_i}\,\mathbf{\Omega}_n \cdot \oiint_{\partial V_i}\psi_n(\mathbf{r})\,\mathbf{n}(\mathbf{r})\,d^2r + \sigma_{t,i}\psi_{n,i} = q_{n,i}\ , \tag{2.41}$$

where $\mathbf{n}(\mathbf{r})$ is the outward pointing unit normal vector on the boundary of the cell. In most cases, the cells of the mesh are polyhedra with planar faces, and hence the surface integral can be expanded in a sum of integrals over each face of the cell, each with a constant $\mathbf{n}(\mathbf{r})$. This allows us to write equation 2.41 as

$$\left(\sum_{f=1}^{F_i} \frac{\mathbf{\Omega}_n \cdot \mathbf{n}_f}{V_i} \iint_{\partial V_{i,f}} \psi_n (\mathbf{r})\, d^2 r \right) + \sigma_{t,i} \psi_{n,i} = q_{n,i} \ , \tag{2.42}$$

where F_i is the number of faces in cell i, and $\mathbf{n}_f$ is the outward pointing unit normal vector on face f. At this point, some freedom is available in choosing how each term in the summation is related to the values of $\psi_{n,i}$ for the cells on either side of face f via interpolation. Once the terms in the summation are expressed in terms of the average values on either side of the face and every cell in the mesh is volume averaged as given above, the result is a linear system of equations whose solution is a vector containing the average value of the flux in each cell of the mesh. Different interpolation schemes will lead to different accuracies for a given problem. In general, the finite volume method works well for conservation equations with smooth solutions, because conservation is preserved within each cell of the mesh, and hence throughout the entire problem. While the transport equation is a conservation equation, its solution is likely to have discontinuities, and this presents difficulties for solving it using a finite volume method.

A common finite volume method used to solve the transport equation is the diamond difference (DD) spatial discretization scheme. To illustrate this method, consider the one dimensional discretized domain shown in Figure 2.4. This figure uses the standard notation in which half integral indices are used at cell edges and integral indices are used at cell centers. Denoting $\mu_n = \Omega_{n,x}$, equation 2.39 for one dimensional problems can be written as

$$\mu_n \frac{\partial \psi_n}{\partial x} + \sigma_t (x)\, \psi_n (x) = q_n (x) \ . \tag{2.43}$$

Volume averaging over cell i in this case is equivalent to simply integrating this equation from $x_{i-1/2}$ to $x_{i+1/2}$, and dividing by $\Delta x_i = x_{i+1/2} - x_{i-1/2}$, resulting in

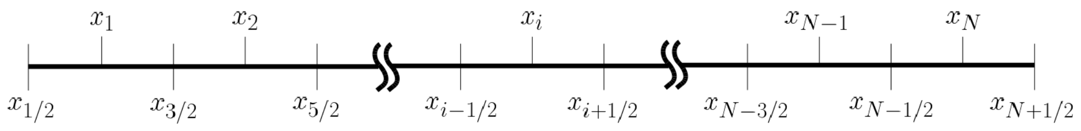


Figure 2.4: One dimensional spatial discretization.

$$\frac{\mu_n}{\Delta x_i}\left(\psi_{n,i+1/2} - \psi_{n,i-1/2}\right) + \sigma_{t,i}\left(\frac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}} \psi_n(x)\,dx\right) = \frac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}} q_n(x)\,dx \ . \tag{2.44}$$

where $\psi_{n,i\pm1/2} = \psi_n\left(x_{i\pm1/2}\right)$ and $\sigma_{t,i}$ is the total cross section in cell i. We then define

$$\psi_{n,i} = \frac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}} \psi_n(x)\,dx \tag{2.45}$$

and

$$q_{n,i} = \frac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}} q_n(x)\,dx \tag{2.46}$$

to rewrite equation 2.44 as

$$\frac{\mu_n}{\Delta x_i}\left(\psi_{n,i+1/2} - \psi_{n,i-1/2}\right) + \sigma_{t,i}\psi_{n,i} = q_{n,i} \ . \tag{2.47}$$

Instead of interpolating between cell averaged angular fluxes to get the cell edge angular fluxes, as is done in most finite volume methods, the DD scheme solves for the cell edge angular fluxes, and linearly interpolates between these values to get the cell averaged angular fluxes. The two approaches can be shown to be equivalent. The interpolation is performed by adding the closing equation

$$\psi_{n,i} = \frac{1}{2}\left(\psi_{n,i-1/2} + \psi_{n,i+1/2}\right) \ . \tag{2.48}$$

2.4.2 Finite Element Methods

The finite element method aims to solve partial differential equations such as equation 2.39 by first converting the differential equation to what is known as the weak form, or variational form. This is accomplished by multiplying the equation by a test function belonging to a certain function space, and then integrating over the spatial domain. To illustrate this, we multiply equation 2.39 by test function $\omega(\mathbf{r}) \in W$, where W is some infinite-dimensional function space, and integrate over the spatial domain

$$\iiint_D \omega(\mathbf{r}) \Big(\mathbf{\Omega}_n \cdot \nabla \psi_n(\mathbf{r}) + \sigma_t(\mathbf{r}) \psi_n(\mathbf{r}) \Big) d^3r = \iiint_D \omega(\mathbf{r}) q_n(\mathbf{r}) d^3r \,. \tag{2.49}$$

The goal now is to find $\psi_n(\mathbf{r}) \in W$ such that equation 2.49 is satisfied for all $\omega(\mathbf{r}) \in W$. If we could find such a $\psi_n(\mathbf{r})$, it can be shown that this function will also be the solution to equation 2.39.[14] No approximations have been made yet, but we also have an infinite-dimensional space of functions for which to satisfy equation 2.49, and this is clearly not ideal. The approximation comes in restricting the function space W to a finite number of dimensions, with the restricted space denoted $\widetilde{W}$. This finite-dimensional function space is typically a space of piece-wise polynomials that are only non-zero in a single cell, or group of cells, in the mesh.

The finite element solution then proceeds by multiplying equation 2.39 by each function in a basis set for the finite space $\widetilde{W}$, integrating over the domain, and expanding $\psi_n(\mathbf{r})$ as a linear combination of the basis functions in $\widetilde{W}$. The result is a linear system of equations whose solution is a vector containing the coefficients of the expansion. The finite element method therefore results in a solution that has a functional form within each cell, rather than simply a piece-wise constant solution matching the average of the solution in each cell as in the finite volume method.

There is also a great deal of freedom in selecting the finite function space $\widetilde{W}$, which may result in higher accuracy than a finite volume method could attain. It is also the case that discontinuous test functions could be used, allowing for discontinuities in the solution, for problems where such discontinuities are likely to exist.

Consider for example the linear discontinuous finite element (LDFE) spatial discretization scheme. To illustrate this method, we will again use the one dimensional discretized domain shown in Figure 2.4. We begin by defining the test functions

$$\omega_i^1(x) = \begin{cases} 1 & \text{for } \; x_{i-1/2} < x < x_{i+1/2} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\omega_i^x(x) = \begin{cases} 6\left(x - x_i\right)/\Delta x_i & \text{for } \; x_{i-1/2} < x < x_{i+1/2} \\ 0 & \text{otherwise} \end{cases}$$

for each cell in the mesh. It is easy to see that these test functions are discontinuous, and hence any solution described by a linear combination of these test functions may also be discontinuous. Since the test functions for cell i are only non-zero within cell i, multiplying the transport equation by each of these test functions and integrating over the problem domain is equivalent to multiplying by the test functions and integrating over the cell. In addition, we divide by the cell width Δx_i in order to express certain quantities as moments of the angular flux. This action results in the following two equations:

$$\frac{\mu_n}{\Delta x_i}\left(\psi_{n,i+1/2} - \psi_{n,i-1/2}\right) + \sigma_{t,i}\psi_{n,i} = q_{n,i} \; , \tag{2.50}$$

$$\frac{3\mu_n}{\Delta x_i}\left(\psi_{n,i+1/2} + \psi_{n,i-1/2} - 2\psi_{n,i}\right) + \sigma_{t,i}\psi_{n,i}^x = q_{n,i}^x \; , \tag{2.51}$$

where $\psi_{n,i\pm1/2}$, $\psi_{n,i}$, and $q_{n,i}$ are defined as in the DD example, and the superscripted variables are defined as

$$\psi^x_{n,i} = \frac{1}{\Delta x_i} \int_{x_{i-1/2}}^{x_{i+1/2}} \omega^x_i\left(x\right) \psi_n\left(x\right) dx \tag{2.52}$$

and

$$q^x_{n,i} = \frac{1}{\Delta x_i} \int_{x_{i-1/2}}^{x_{i+1/2}} \omega^x_i\left(x\right) q_n\left(x\right) dx \ . \tag{2.53}$$

At this point, we must choose how to handle the boundary information. We choose the upwind scheme in which the flux on the edge entering the cell in direction $\mathbf{\Omega}_n$ is specified by the flux in the upwind cell. In this simple one dimensional example, this means that for $\mu_n > 0$, $\psi_{n,i-1/2}$ is known, and for $\mu_n < 0$, $\psi_{n,i+1/2}$ is known. For brevity, we restrict this example to the case where $\mu_n > 0$. In this case, there are three unknowns in equations 2.50 and 2.51: $\psi_{n,i}$, $\psi^x_{n,i}$, and $\psi_{n,i+1/2}$. The next step is to express $\psi_n\left(x\right)$ as a linear combination of the test functions

$$\psi_n\left(x\right) = \psi_{n,i} + \frac{2\left(x - x_i\right)}{\Delta x_i} \psi^x_{n,i} \ . \tag{2.54}$$

It is easy to show that with this expansion, equations 2.45 and 2.52 are automatically satisfied. We can now use this expansion to evaluate the angular flux at the right boundary as

$$\psi_{n,i+1/2} = \psi_{n,i} + \psi^x_{n,i} \ . \tag{2.55}$$

Plugging this result into equations 2.50 and 2.51 results in a system of two equations and two unknowns, $\psi_{n,i}$ and $\psi^x_{n,i}$, for each cell i in the spatial mesh. The final result is a lower triangular matrix equation consisting of two equations per spatial mesh cell, for the $\mathbf{\Omega}_n$ direction.

2.4.3 Method of Characteristics

The method of characteristics (MOC) was developed by Askew[15] as an integral method to avoid negativities in the numerical angular flux solution, as well as to handle complex geometrical models. Implementations of the MOC lie in two categories; the method of long characteristics and the method of short characteristics. In the long characteristic method, the transport equation is solved analytically along lines drawn through the entire spatial domain via the use of an integrating factor. As each line crosses through a spatial cell, the particle source and material properties are often assumed constant, and the average angular flux along the line segment is computed. A volume is then associated with this line segment so that the contribution of the angular flux to the scalar flux within the spatial cell can be computed. The short characteristic method on the other hand treats each spatial cell individually, as was done in the finite volume and finite element methods, with the solution in each cell along short characteristic lines spanning the spatial cell, given analytically via the use of an integrating factor.

The first step in any characteristic method is to make the characteristic transformation to the transport equation. This is done by parameterizing the position variable with respect to a reference position $\mathbf{r_0}$ by defining $\mathbf{r} = \mathbf{r_0} + s\boldsymbol{\Omega}_n$. The streaming term in the transport equation then becomes

$$\begin{aligned}\boldsymbol{\Omega}_n \cdot \nabla \psi_n (\mathbf{r}) = \Omega_{n,x} \frac{\partial \psi_n}{\partial x} + \Omega_{n,y} \frac{\partial \psi_n}{\partial y} + \Omega_{n,z} \frac{\partial \psi_n}{\partial z} &= \\ \frac{dx}{ds}\frac{\partial \psi_n}{\partial x} + \frac{dy}{ds}\frac{\partial \psi_n}{\partial y} + \frac{dz}{ds}\frac{\partial \psi_n}{\partial z} &= \frac{d}{ds}\psi_n (\mathbf{r}) \ . \qquad (2.56)\end{aligned}$$

Equation 2.39 can then be written as

$$\frac{d}{ds}\psi_n\left(\mathbf{r_0}+s\mathbf{\Omega}_n\right)+\sigma_t\left(\mathbf{r_0}+s\mathbf{\Omega}_n\right)\psi_n\left(\mathbf{r_0}+s\mathbf{\Omega}_n\right)=q_n\left(\mathbf{r_0}+s\mathbf{\Omega}_n\right)\ . \tag{2.57}$$

To simplify this further, we switch to a one dimensional coordinate system where $\mathbf{r_0}$ is at the origin, and the axis is in the direction of $\mathbf{\Omega}_n$ so that we can rewrite equation 2.57 as

$$\frac{d}{ds}\psi_n\left(s\right)+\sigma_t\left(s\right)\psi_n\left(s\right)=q_n\left(s\right)\ . \tag{2.58}$$

This equation can then be solved analytically through the use of the integrating factor $e^{-\int_0^s \sigma_t(s')ds'}$. The final result is

$$\psi_n\left(\mathbf{r_0}+s\mathbf{\Omega}_n\right)=\psi_n\left(\mathbf{r_0}\right)e^{-\int_0^s \sigma_t(s')ds'}+\int_0^s q_n\left(\mathbf{r_0}+s''\mathbf{\Omega}_n\right)e^{-\int_{s''}^s \sigma_t(s')ds'}ds''\ . \tag{2.59}$$

It is important to note that up until this point, no approximations have been made. Equation 2.59 is the exact solution to equation 2.39 along a given characteristic line. Of course, the solution along a single line is not what is sought in most cases. Approximations are made in combining the solutions along a large number of lines to get the solution to equation 2.39 throughout the entire spatial domain. The more lines used in the calculation, the more accurate the numerical solution will be. It is also the case that sources and cross sections are assumed constant within spatial cells of a mesh, and hence the accuracy of the calculation is also tied directly to the resolution of the mesh. Finally, it is clear from equation 2.59 that as long as the source function $q_n\left(\mathbf{r}\right)$ is strictly positive, the solution will be as well. This is an important quality, since finite volume and finite element methods are prone to producing negative angular fluxes in optically thick cells.

2.5 The Slice Balance Approach

The Slice Balance Approach (SBA) is yet another spatial discretization method that warrants further discussion due to its relevance to the research presented here. The SBA is a characteristic based, multiple balance scheme for solving the discrete ordinates equations on unstructured meshes developed by Grove.[9] The multiple balance scheme was defined by Morel and Larsen[16], where they state:

> "We define any S_N differencing scheme based on the use of whole-cell and approximate subcell balance equations as a multiple balance scheme. Note that this is a very general form of definition. This approach can clearly be applied to any form of the transport equation. Furthermore, there are infinitely many multiple balance schemes that can be defined for the same form of the transport equation and the same phase-space cell."

The SBA solves for the angular flux in each spatial cell by first decomposing the cell into what are called slices. A slice is the union of all points in a cell for which a line drawn through the point in the discrete ordinate direction $\mathbf{\Omega}_n$, intersects the same inlet and outlet face of the cell. This is illustrated in Figure 2.5, which identifies slice ij formed from inlet face i and outlet face j of a polyhedral spatial cell with 12 faces. There are a total of 17 slices that can be made from the cell shown in Figure 2.5. Since the slices within a cell are non-overlapping, and their union is the cell itself, summing the volume integrated transport equation for all slices equals the volume integrated transport equation for the cell. The goal of SBA is then to solve for the flux within each slice and the flux on the outgoing boundary of each slice, and use these quantities to solve for the flux within the cell and the flux on the cell's outgoing faces.

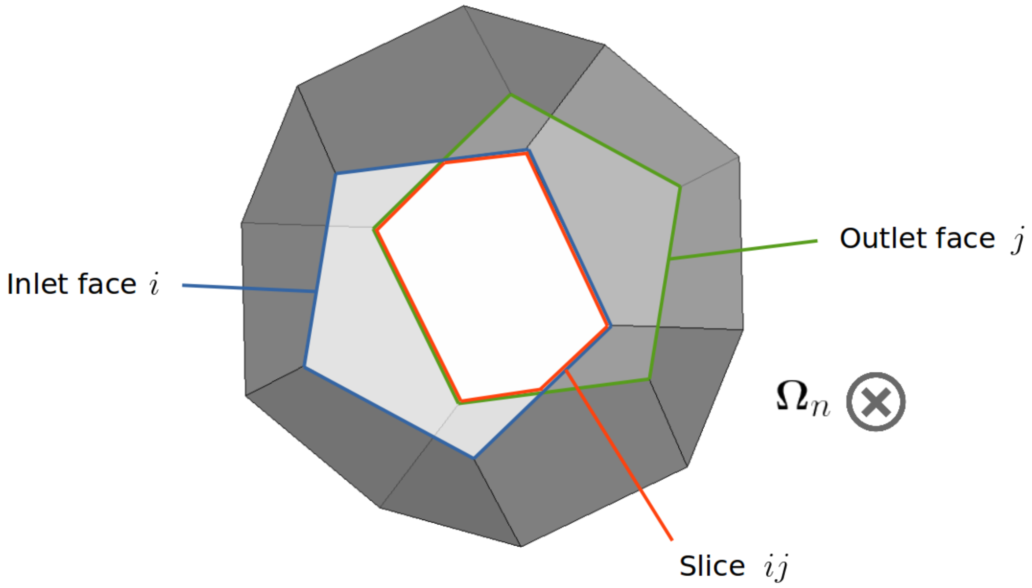


Figure 2.5: Illustration of slice ij formed from inlet face i and outlet face j of a polyhedral spatial cell.

To derive the SBA framework, consider the discrete ordinates transport equation integrated over a polyhedral cell c with planar faces

$$\sum_{i=1}^{F_i} \mathbf{\Omega}_n \cdot \mathbf{n}_i A_i \psi_{n,i} + \sum_{j=1}^{F_j} \mathbf{\Omega}_n \cdot \mathbf{n}_j A_j \psi_{n,j} + \sigma_{t,c} V_c \, \psi_{n,c} = V_c \, q_{n,c} \ , \tag{2.60}$$

where

$$\psi_{n,i} = \frac{1}{A_i} \iint_{\partial V_i} \psi_n \left(\mathbf{r}\right) d^2 r \ ,$$

$$\psi_{n,j} = \frac{1}{A_j} \iint_{\partial V_j} \psi_n \left(\mathbf{r}\right) d^2 r \ ,$$

$$\psi_{n,c} = \frac{1}{V_c} \iiint_{V_c} \psi_n \left(\mathbf{r}\right) d^3 r \ ,$$

$$q_{n,c} = \frac{1}{V_c} \iiint_{V_c} q_n \left(\mathbf{r}\right) d^3 r \ ,$$

i is an index used for inlet faces, j is an index used for outlet faces, A is an area, F is the number of faces of each type, $\mathbf{n}$ is an outward pointing unit normal vector, and V is a volume. The unknowns appearing in this equation are $\psi_{n,j}$ and $\psi_{n,c}$ since the $\psi_{n,i}$ are given via boundary conditions or the upstream cell. Further, we define

$$\alpha_{n,i} = \mathbf{\Omega}_n \cdot \mathbf{n}_i A_i \quad \text{and} \quad \alpha_{n,j} = \mathbf{\Omega}_n \cdot \mathbf{n}_j A_j$$

to rewrite equation 2.60 as

$$\sum_{i=1}^{F_i} \alpha_{n,i}\psi_{n,i} + \sum_{j=1}^{F_j} \alpha_{n,j}\psi_{n,j} + \sigma_{t,c}V_c\,\psi_{n,c} = V_c\,q_{n,c}\ . \tag{2.61}$$

Similarly integrating the transport equation over slice s yields

$$\alpha_{n,\text{in}}\psi_{n,s,\text{in}} + \alpha_{n,\text{out}}\psi_{n,s,\text{out}} + \sigma_{t,c}V_s\,\psi_{n,s} = V_s\,q_{n,s}\ , \tag{2.62}$$

where each term is defined analogously to the terms appearing in equation 2.61. The important thing to notice here is that each sum appearing in equation 2.61 has reduced to a single term, since there are now only two faces for which $\mathbf{\Omega}_n \cdot \mathbf{n} \neq 0$. Obtaining the solution to equation 2.62 for each slice in the cell then allows the reconstruction of the cell unknowns via the following two relations

$$V_c\,\psi_{n,c} = \sum_s V_s\,\psi_{n,s}\ , \tag{2.63}$$

$$A_j\,\psi_{n,j} = \sum_{s*j} A_{s,\text{out}}\,\psi_{n,s,\text{out}}\ , \tag{2.64}$$

where the summation over $s*j$ denotes a sum over all slices with outgoing faces that are partial faces of cell outgoing face j.

Solving equation 2.62 for each slice in Figure 2.5 results in 17 relatively simple problems, each with two unknowns, instead of solving the more complicated single problem for the cell with 7 unknowns. While more work must be done due to the increase in the number of unknowns being solved for, there are some fairly significant benefits. First, this method provides a general way to treat unstructured arbitrary polyhedral meshes. In addition, equation 2.62 resembles the one dimensional volume integrated transport equation encountered in the examples of the DD and LDFEM schemes given previously, hinting that one dimensional spatial discretization schemes can be applied to the three dimensional problem.

Second, the accuracy of the solution compared to the traditional cell balance approach (CBA) is likely to increase due to two factors. The first factor is the increased spatial resolution on which the solution is obtained, even if this solution is then coarsened to represent the solution on the cells of the mesh rather than the slices. The second factor is that the SBA leads to a more consistent representation of particle streaming than the traditional CBA. To see why this is the case, consider Figures 2.6 and 2.7, which show the two dimensional comparison between the solution on a quadrilateral cell using the CBA and SBA respectively. In these figures, ψ_L, ψ_R, ψ_B, and ψ_T are the fluxes on the left, right, bottom, and top edges of the cell respectively, and ψ_k, $\psi_{k,\text{out}}$, and $l_{k,\text{out}}$ are the average flux in slice k, the average flux on the outlet edge of slice k, and the width of the outlet edge of slice k respectively. In the CBA, ψ_R is a function of ψ_L. However, given $\boldsymbol{\Omega}_n$ as oriented in these figures, this should not be the case, as particles cannot stream from the left edge to the right edge. With the SBA, this non-physical causality is avoided, and ψ_R is given as the outlet flux of Slice 3, whose inlet flux is equal to ψ_B. This is reminiscent of characteristic methods which use this sort of cell decomposition implicitly, while the SBA uses it explicitly.

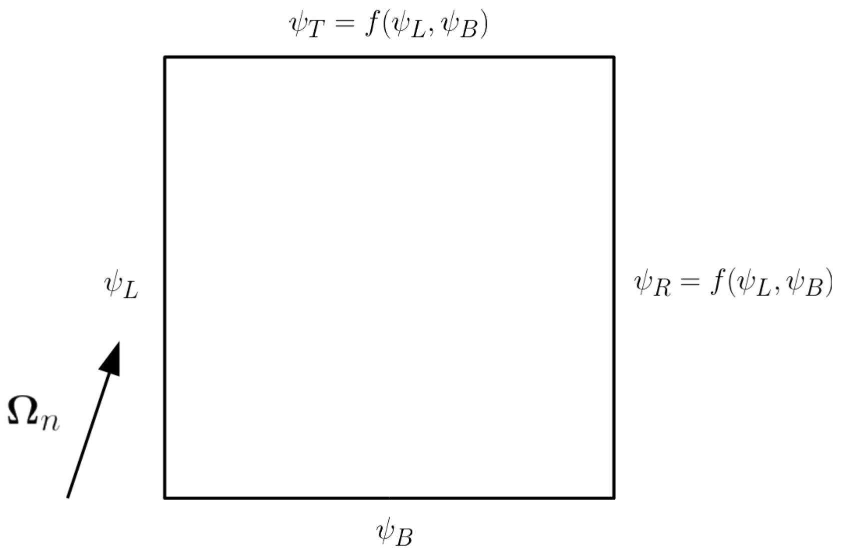


Figure 2.6: A single quadrilateral cell and the relationship between the flux variables using the CBA.

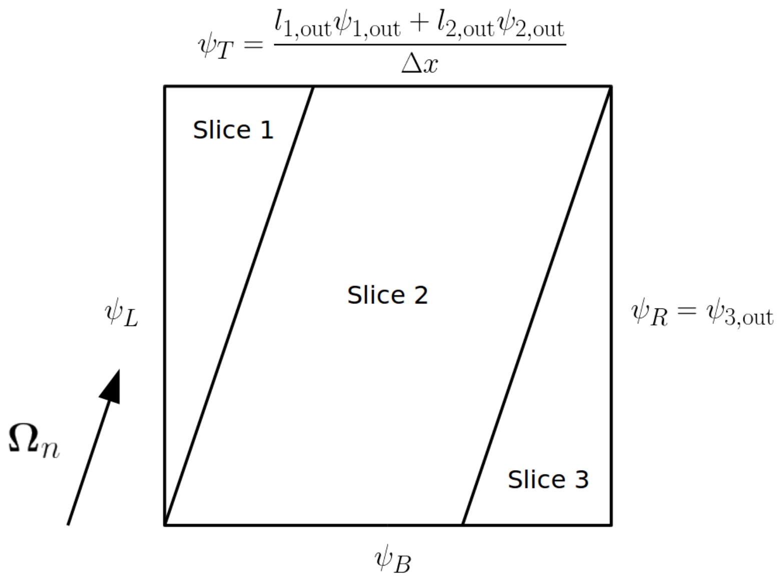

$\psi_T = \frac{l_{1,\text{out}}\psi_{1,\text{out}} + l_{2,\text{out}}\psi_{2,\text{out}}}{\Delta x}$
Slice 1
Slice 2
Slice 3
ψ_L
$\psi_R = \psi_{3,\text{out}}$
$\mathbf{\Omega}_n$
ψ_B

Figure 2.7: A single quadrilateral cell and the relationship between the flux variables using the SBA.

2.6 Iterative Methods

In the section discussing energy discretization, the Gauss-Seidel iterative method was briefly discussed. Mathematically, the Gauss-Seidel method can be written as

$$\begin{aligned}
\mathbf{\Omega} \cdot \nabla \psi_g^{(k+1)}(\mathbf{r}, \mathbf{\Omega}) + \sigma_{t,g}(\mathbf{r})\, \psi_g^{(k+1)}(\mathbf{r}, \mathbf{\Omega}) = \\
\sum_{l=0}^{L_1} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s,g\to g}(\mathbf{r})\, \phi_{l,g}^{m(k+1)}(\mathbf{r})\, Y_l^m(\mathbf{\Omega}) + \\
\sum_{g'=1}^{g-1} \sum_{l=0}^{L_1} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s,g'\to g}(\mathbf{r})\, \phi_{l,g'}^{m(k+1)}(\mathbf{r})\, Y_l^m(\mathbf{\Omega}) + \\
\sum_{g'=g+1}^{G} \sum_{l=0}^{L_1} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s,g'\to g}(\mathbf{r})\, \phi_{l,g'}^{m(k)}(\mathbf{r})\, Y_l^m(\mathbf{\Omega}) + \\
\sum_{l=0}^{L_2} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} q_{l,g}^m(\mathbf{r})\, Y_l^m(\mathbf{\Omega}) \ , \quad (2.65)
\end{aligned}$$

where k is the iteration index. We can define

$$\begin{aligned}
Q_g(\mathbf{r}, \mathbf{\Omega}) = \sum_{g'=1}^{g-1} \sum_{l=0}^{L_1} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s,g'\to g}(\mathbf{r})\, \phi_{l,g'}^{m(k+1)}(\mathbf{r})\, Y_l^m(\mathbf{\Omega}) + \\
\sum_{g'=g+1}^{G} \sum_{l=0}^{L_1} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s,g'\to g}(\mathbf{r})\, \phi_{l,g'}^{m(k)}(\mathbf{r})\, Y_l^m(\mathbf{\Omega}) + \\
\sum_{l=0}^{L_2} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} q_{l,g}^m(\mathbf{r})\, Y_l^m(\mathbf{\Omega}) \quad (2.66)
\end{aligned}$$

in order to rewrite equation 2.65 as

$$\mathbf{\Omega} \cdot \nabla \psi_g^{(k+1)}(\mathbf{r},\mathbf{\Omega}) + \sigma_{t,g}(\mathbf{r})\,\psi_g^{(k+1)}(\mathbf{r},\mathbf{\Omega}) = \sum_{l=0}^{L_1}\sum_{m=-l}^{+l}\frac{2l+1}{4\pi}\sigma_{l,s,g\to g}(\mathbf{r})\,\phi_{l,g}^{m(k+1)}(\mathbf{r})\,Y_l^m(\mathbf{\Omega}) + Q_g(\mathbf{r},\mathbf{\Omega}) \ . \tag{2.67}$$

Each iteration of the Gauss-Seidel method requires the solution to equation 2.67 for $g = 1, 2, \ldots, G$. The important thing to notice is that all coupling between energy groups is now contained in the term $Q_g(\mathbf{r},\mathbf{\Omega})$. This term can be computed prior to solving equation 2.67 using angular flux moments $\phi_{l,g'}^m(\mathbf{r})$ from either the current iteration if $g' < g$, or the previous iteration if $g' > g$. After computing $Q_g(\mathbf{r},\mathbf{\Omega})$, equation 2.67 is a steady state, energy independent transport equation for $\psi_g^{(k+1)}(\mathbf{r},\mathbf{\Omega})$. The result is that the solution for each group g must be obtained to a transport equation of the form

$$\mathbf{\Omega} \cdot \nabla \psi(\mathbf{r},\mathbf{\Omega}) + \sigma_t(\mathbf{r})\,\psi(\mathbf{r},\mathbf{\Omega}) = \sum_{l=0}^{L_1}\sum_{m=-l}^{+l}\frac{2l+1}{4\pi}\sigma_{l,s}(\mathbf{r})\,\phi_l^m(\mathbf{r})\,Y_l^m(\mathbf{\Omega}) + Q(\mathbf{r},\mathbf{\Omega}) \ . \tag{2.68}$$

Equation 2.68 requires its own iterative method to handle the within group scattering term on the right hand side. Source iteration is a particularly simple and intuitive method whereby the scattering term is evaluated from the previous iteration, or an initial guess on the first iteration. With an initial guess of $\psi^{(0)}(\mathbf{r},\mathbf{\Omega}) = 0$, each iteration computes the angular flux due to particles which have scattered k times, where k is again the iteration index. Mathematically, source iteration can be written as

$$\mathbf{\Omega} \cdot \nabla \psi^{(k+1)}(\mathbf{r}, \mathbf{\Omega}) + \sigma_t(\mathbf{r}) \psi^{(k+1)}(\mathbf{r}, \mathbf{\Omega}) = \sum_{l=0}^{L_1} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s}(\mathbf{r}) \phi_l^{m(k)}(\mathbf{r}) Y_l^m(\mathbf{\Omega}) + Q(\mathbf{r}, \mathbf{\Omega}) \ . \quad (2.69)$$

Since all quantities on the right hand side are computed using the previous iteration, equation 2.69 is a fixed source, steady state, energy independent transport equation for $\psi^{(k+1)}(\mathbf{r}, \mathbf{\Omega})$. The result is that the solution for each iteration must be obtained to a transport equation of the form

$$\mathbf{\Omega} \cdot \nabla \psi(\mathbf{r}, \mathbf{\Omega}) + \sigma_t(\mathbf{r}) \psi(\mathbf{r}, \mathbf{\Omega}) = q(\mathbf{r}, \mathbf{\Omega}) \ , \quad (2.70)$$

which was our starting point for discussing spatial discretization schemes.

The source iteration method, while simple and intuitive, is rarely used in transport codes due to its propensity to converge very slowly in problems for which particles scatter many times before absorption or leakage. Instead, efforts are made to accelerate the convergence. One such acceleration method is diffusion synthetic acceleration (DSA) in which each iteration consists of two steps. The first step is to estimate the angular flux given the flux moments of the previous iteration, as was done in source iteration. The second step is to estimate the error in the scalar flux via the use of a diffusion operator, and to use this error to correct the scalar flux estimate. Mathematically, DSA can be written as

$$\mathbf{\Omega} \cdot \nabla \psi^{(k+1/2)}(\mathbf{r}, \mathbf{\Omega}) + \sigma_t(\mathbf{r}) \psi^{(k+1/2)}(\mathbf{r}, \mathbf{\Omega}) = \sum_{l=0}^{L_1} \sum_{m=-l}^{+l} \frac{2l+1}{4\pi} \sigma_{l,s}(\mathbf{r}) \phi_l^{m(k)}(\mathbf{r}) Y_l^m(\mathbf{\Omega}) + Q(\mathbf{r}, \mathbf{\Omega}) \ , \quad (2.71)$$

$$\phi^{(k+1/2)}(\mathbf{r}) = \iint_{4\pi} \psi^{(k+1/2)}(\mathbf{r}, \mathbf{\Omega})\, d\Omega \ , \tag{2.72}$$

$$-\nabla \cdot \left(\frac{1}{3\sigma_{tr}(\mathbf{r})} \nabla \delta\phi^{(k+1/2)}(\mathbf{r}) \right) + \sigma_a(\mathbf{r})\, \delta\phi^{(k+1/2)}(\mathbf{r}) = \sigma_s(\mathbf{r}) \left(\phi^{(k+1/2)}(\mathbf{r}) - \phi^{(k)}(\mathbf{r}) \right) \ , \tag{2.73}$$

$$\phi^{(k+1)}(\mathbf{r}) = \phi^{(k+1/2)}(\mathbf{r}) + \delta\phi^{(k+1/2)}(\mathbf{r}) \ , \tag{2.74}$$

where σ_a is the absorption cross section, σ_s is the scattering cross section, $\sigma_{tr} = \sigma_t - \overline{\mu}\,\sigma_s$, and $\overline{\mu}$ is the average scattering angle cosine. For isotropic, or weakly anisotropic scattering in the lab frame, and a consistent spatial discretization for both the transport and diffusion operators, DSA is an unconditionally effective way to accelerate the source iteration method. DSA is a special case of the angular multi-grid strategy, where the diffusion operator represents the coarse grid solution used to attenuate low frequency errors. It is effective because it is the low frequency errors that the transport operator is ineffective at converging. Where source iteration has a spectral radius equal to $c = \sigma_s/\sigma_t$, DSA if implemented consistently for problems with isotropic scattering can reduce this to roughly $0.23c$.

While many other iterative methods exist, the small subset discussed above share a common feature that each inner iteration consists of the solution to the fixed source, steady state, energy independent transport equation for the entire spatial domain using previous-iterate values for the volumetric source information. They essentially decompose a time dependent, energy dependent, and angle dependent problem into a series of problems that have only spatial dependence. A desirable property of such methods is that the iteration counts do not change with spatial mesh refinement, which is an important consideration for high resolution transport problems.[17]

2.7 The Parallel Transport Sweep

With iterative methods such as source iteration and DSA, which converge in roughly the same number of iterations regardless of the spatial mesh resolution, the time to solution is largely determined by the time required for each inner iteration to solve the fixed source, steady state, energy independent transport equation for the entire spatial domain. This places a considerable amount of emphasis on reducing the time per inner iteration by parallelizing the solution on multiple CPUs.

Given the memory requirements to store the angular flux solution, high fidelity transport calculations are carried out on super-computers and clusters, with each node of the machine storing the solution on a subset of the problem geometry. The solution to equation 2.39 is then obtained through what is known as a transport sweep. The process starts by obtaining the solution on a subset of the geometry for an angle for which all incoming fluxes are given by the prescribed boundary conditions (typically a subset in the corner of the spatial domain). After this node has obtained the solution for this angle, it sends messages to the neighboring nodes so that they will receive the necessary boundary information to solve for this angle as well. In this way, the sweep progresses in a plane that has one fewer dimension than the problem geometry.

The transport sweep is illustrated in Figure 2.8 for a two dimensional domain with each square representing a subset of the mesh owned by a single node of the machine and each arrow representing communication between nodes. In this illustration, four sweep planes are progressing simultaneously, each starting from a corner of the spatial domain, and each using a different color for its arrows. The subsets colored in gray denote angular collisions where a node has received boundary information for multiple angles, and must decide which calculation to perform first. In reality, as soon as the corner subset completes the first angle and sends its outgoing fluxes to its neighbors, it would begin solving the next angle in the quadrature set for which all incoming fluxes are given by the prescribed boundary conditions, and then the next angle after that and so on as shown in Figure 2.9. In this way, once the initial angle swept reaches the inner most node, all nodes will be busy performing calculations until there are no more angles to sweep. The parallel efficiency of the sweep is largely determined by how long all nodes can be kept busy performing calculations as opposed to waiting on boundary information to begin performing calculations.

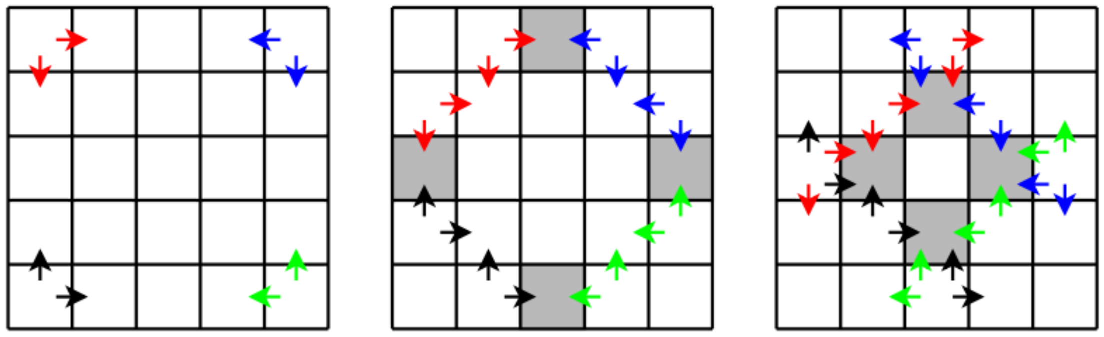


Figure 2.8: Two dimensional illustration of the parallel transport sweep for four angles emanating from the four corners of the spatial domain.

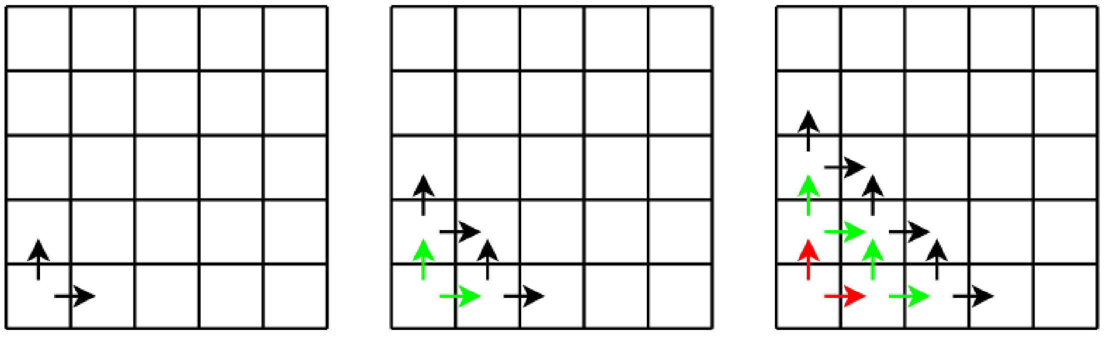


Figure 2.9: Two dimensional illustration of the parallel transport sweep for three angles emanating from the same corner of the spatial domain.

An alternative to the transport sweep is the parallel block Jacobi (PBJ) method which is an iterative method whereby all nodes are actively performing calculations using boundary information from the previous iteration. While this method has the obvious advantage of exhibiting zero idle time and no angular collisions, the PBJ method does not have the property that iteration counts for scattering iteration methods are independent of mesh resolution.[18] In addition, despite the idle time inherent in the parallel transport sweep, it has been shown to be scalable to over 10^6 cores.[19]

Aside from the inherent idle time associated with parallel transport sweeps, there is another issue that arises in the case of arbitrary polyhedral meshes, or even tetrahedral meshes. This issue is that inter-node domain boundaries should be kept planar to avoid ray re-entry. Consider Figure 2.10 for instance, which shows a jagged inter-node domain boundary and an angle of the S_N quadrature set, for a two dimensional triangular mesh. During a sweep, node 1 would pass flux information to node 2, but in order to compute the solution on the entirety of node 1, input flux information is needed from node 2. In other words, node 2 is dependent on node 1, and node 1 is dependent on node 2. This complicates the sweep dependency graph and requires further iteration within the parallel transport sweep. While work has been done to remedy this issue [20][21], it is still preferable to have planar inter-node domain boundaries. Unfortunately, such planar inter-node boundaries are not guaranteed to exist in the problem geometry, and must be artificially introduced. Furthermore, it is desired to have roughly the same number of cells in each subset in the interest of load balancing, so such planar inter-node boundaries cannot be placed just anywhere. Recent work in load-balancing extruded triangular meshes has shown that introduction of such inter-node domain boundaries in two dimensions can be a very challenging task.[22]

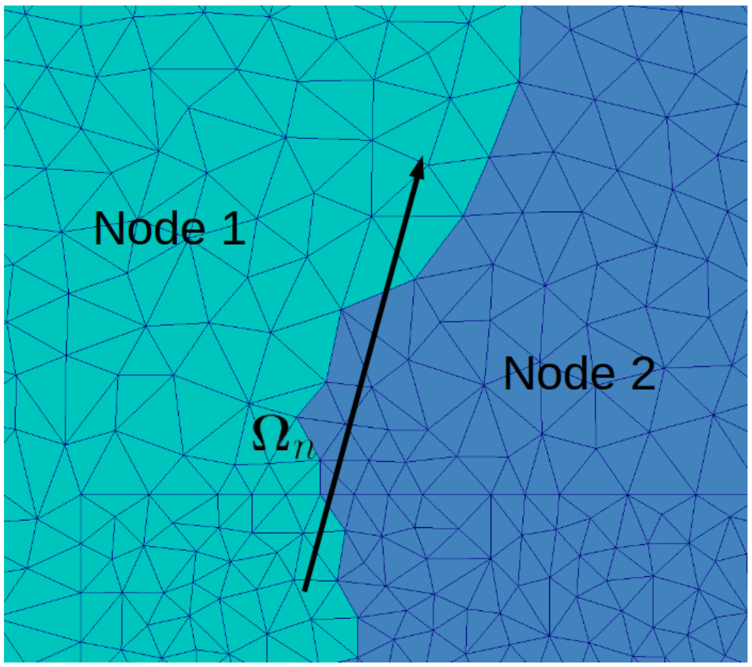


Figure 2.10: Two dimensional triangular mesh with a jagged inter-node boundary.

2.8 Overview

This chapter has discussed a few of the most common deterministic methods for solving the transport equation. A more thorough review can be found in the book by Lewis and Miller[2], and an extensive review of iterative methods for particle scattering can be found in the Progress in Nuclear Energy article by Adams and Larsen.[17] To summarize, the full solution to equation 2.1 is obtained by discretizing the entire phase space of the angular flux. Discretization of the temporal variable results in a series for steady state problems to solve, while iterative methods result in a series of fixed source, energy independent problems to solve. After applying the discrete ordinates approximation, the inner-most problem to solve has only spatial dependence for each angle in the angular quadrature set, and the solution to this problem is parallelized to take advantage of super-computers and clusters for high fidelity transport calculations. This research is particularly focused on the solution to this inner-most problem, and the accuracy and parallel efficiency of its implementation.

3. DERIVATION OF AN EXTENDED SLICE BALANCE APPROACH

The Slice Balance Approach (SBA) presented in the previous chapter was the starting point for the research presented here, which began as an attempt to implement the linear discontinuous finite element (LDFE) spatial discretization into the SBA framework, which was outlined by Kennedy, Watson, and Grove in 2010[23], but never implemented as of 2016. Along the path to such an implementation it was discovered that with very little added cost, a gain in accuracy could be achieved via a simple modification that fundamentally changes how facial angular flux variables are computed and communicated to downstream cells. In addition to describing this modification and the implementation of the LDFE spatial discretization in the extended SBA framework, this chapter also describes a local face-based transport sweep algorithm which solves for the angular flux throughout the spatial domain on a single node for a given discrete ordinate.

3.1 Modifying the Traditional Slice Balance Approach

As discussed in the previous chapter, the SBA gains accuracy through a more consistent representation of particle streaming in a way reminiscent of the Method of Characteristics (MOC). Consider for example, the sliced quadrilateral cell shown in Figure 3.1. In a cell balance method, the flux in the cell interior and the fluxes exiting on the right and top edges would all be influenced by the fluxes entering on the left and bottom edges. However, given $\mathbf{\Omega}_n$ as oriented in this figure, the flux entering on the left edge should not influence the flux exiting on the right edge, because particles cannot stream from the left edge to the right edge. With the SBA, this non-physical causality is avoided.

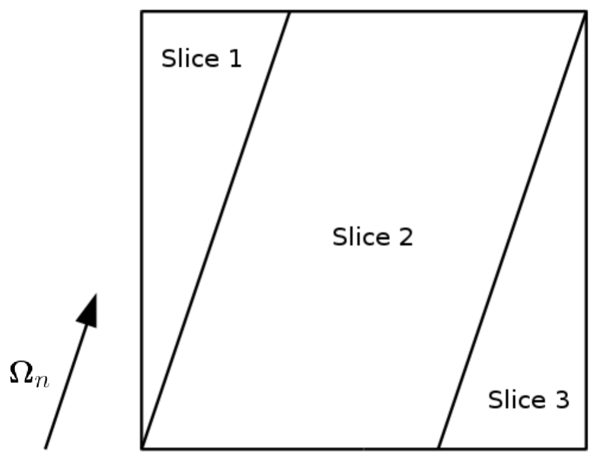


Figure 3.1: Example of a sliced quadrilateral cell.

The SBA solves for the interior and exiting fluxes of each slice using a prescribed spatial discretization scheme, and uses these slice-wise fluxes to construct the interior and exiting cell-wise fluxes. In the example shown in Figure 3.1, the cell interior flux would be constructed from the interior fluxes of the three slices, the flux exiting the cell on the top face would be constructed from the exiting fluxes of Slices 1 and 2, and the flux exiting the cell on the right face would be the exiting flux of Slice 3.

On a per-cell basis, the non-physical causality noted above is avoided by the SBA; however, on a per-face basis it is not. To see why, consider an identical cell to that in Figure 3.1 placed on top of the depicted cell, composed of Slices $1'$, $2'$, and $3'$ as shown in Figure 3.2, ignoring the dotted line for the moment. In this new cell, the flux entering on the bottom face is given by the flux exiting the top face of the original cell, which was constructed from Slices 1 and 2. In the SBA, this flux would be used as the incoming fluxes for Slices $2'$ and $3'$. Physically however, the flux leaving Slice 1 should not influence the flux entering Slice $3'$ because particles cannot stream from Slice 1 to Slice $3'$.

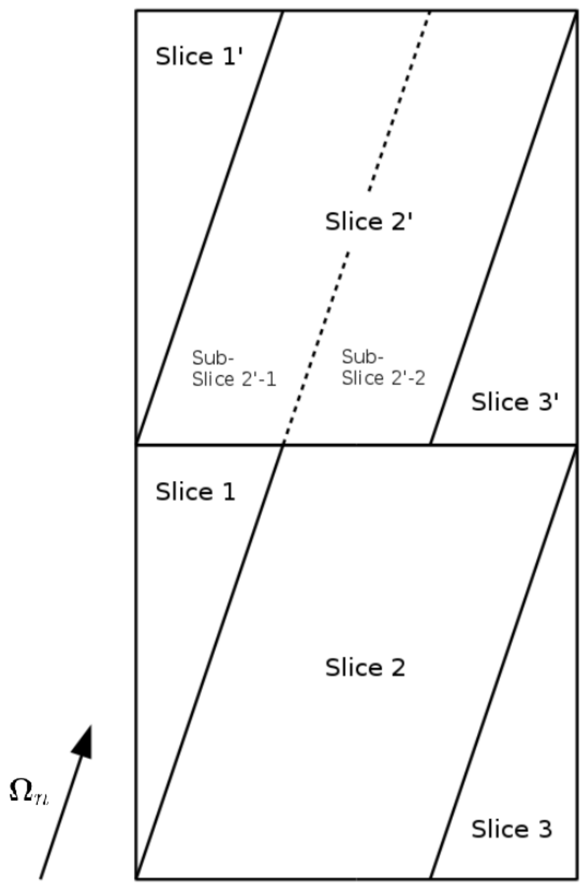


Figure 3.2: Example of two adjacent cells illustrating the concept of a sub-slice.

It is this smearing of the facial fluxes that we attempt to reduce in the proposed modification to the SBA by exploiting the concept of a sub-slice. Where a slice is defined as the cell region bounded by a single inlet-outlet face pair, a sub-slice is defined as the portion of a slice that is downstream of a single slice in the upstream cell, as shown in Figure 3.2. While the streaming plus collision operator is still inverted on each slice, and the cell interior flux is still constructed from the slice fluxes, the cell facial fluxes are not. The incoming fluxes are stored on the sub-slices, and the slice can be treated only after all of its contained sub-slices have received their incoming flux information from their upstream slices. At this point the sub-slice incoming fluxes are appropriately averaged for use in the parent slice. This should further improve accuracy in problems exhibiting shadow type discontinuities, where the flux may be discontinuous in neighboring slices. An example of such a problem is the propagation of a single ray of particles.

The careful observer will note that this alteration to the SBA only serves to propagate discontinuities into the cell immediately downstream, and further non-physical causalities are indeed still present. For instance, placing a third cell on top of the cells in Figure 3.2 composed of Slices $1''$, $2''$, and $3''$, we can see that the flux entering Sub-Slice $2''$-2 would be determined by the flux exiting Slice $2'$, which was influenced by the exiting fluxes of Slices 1 and 2. However, the flux exiting Slice 2 should not influence the incoming flux in Sub-Slice $2''$-2. It should also be noted that propagating discontinuities throughout the entire mesh in this fashion would result in a much more computationally cumbersome beast than either the MOC or traditional cell balance methods. One goal of this research is to determine what effect this single cell downstream discontinuity propagation has on the numerical solution with relatively little added cost to the traditional SBA.

Finally, consider if we were to draw cut planes parallel to $\mathbf{\Omega}_n$ through the spatial mesh as shown in Figure 3.3 and refine the definition of a slice such that no slice straddles a cut plane as shown in Figure 3.4. It is easy to see that the solution in between consecutive cut planes would be independent of the solution in any other such region. This will be an important result in the next chapter, and it is made possible by the concept of the sub-slice, since facial fluxes (for faces which would straddle the cut planes) are no longer needed to begin solving within a slice, making each region between consecutive cut planes completely decoupled from all other such regions.

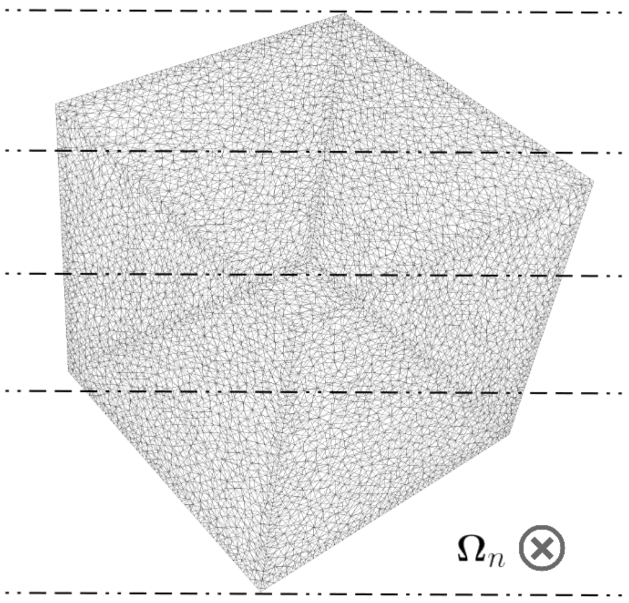


Figure 3.3: Decomposition of a meshed cubic spatial domain by five cut planes.

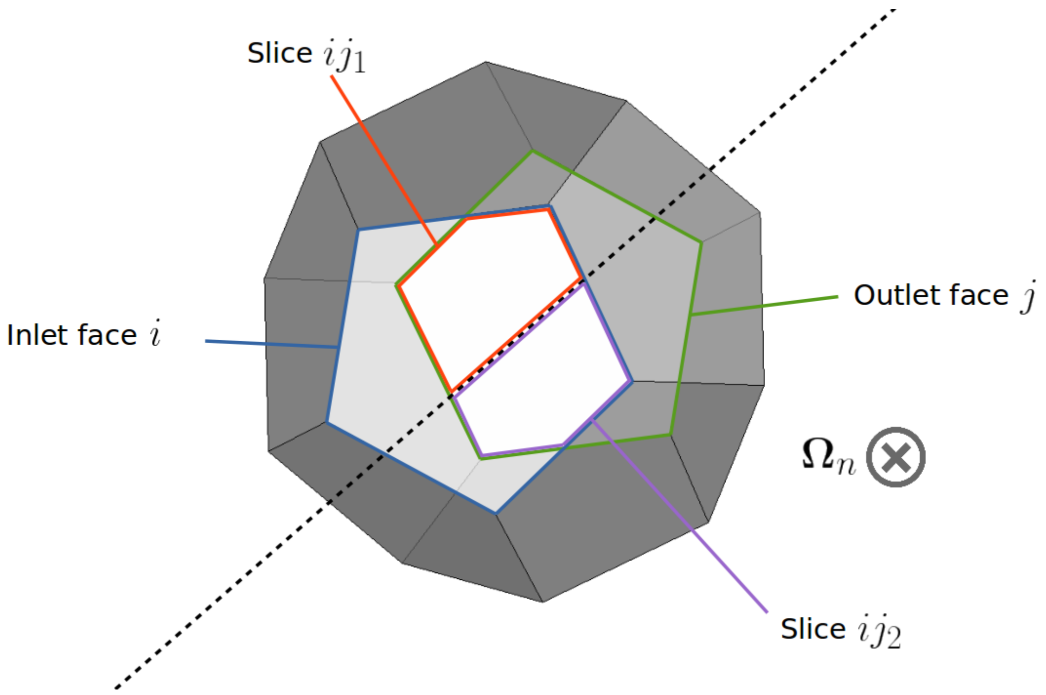


Figure 3.4: Re-definition of a slice such that it does not straddle a cut plane.

3.2 Implementation of the Linear Discontinuous Finite Element Method

To illustrate how the LDFE spatial discretization scheme can be implemented in the extended SBA framework, consider Figure 3.5 which singles out a single slice of a spatial cell. By definition, the slice is aligned with the discrete ordinate $\mathbf{\Omega}_n$, and hence the only faces that are not parallel to the discrete ordinate are the faces labeled $\partial V_{s,\text{in}}$ and $\partial V_{s,\text{out}}$, where the subscript s is used to denote quantities pertaining to slice s. These two surfaces have outward pointing unit vectors labeled $\mathbf{n}_{s,\text{in}}$ and $\mathbf{n}_{s,\text{out}}$ respectively.

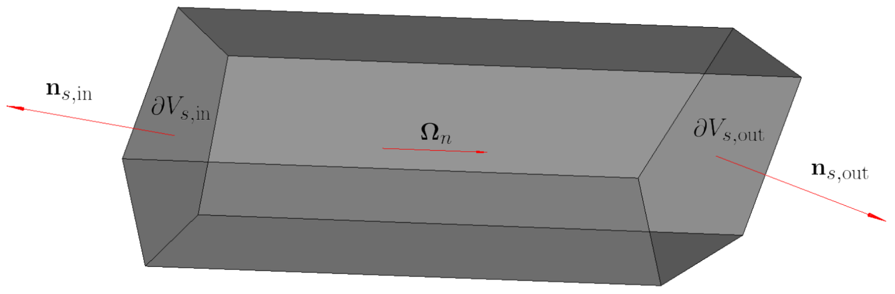

$\mathbf{n}_{s,\text{in}}$
$\partial V_{s,\text{in}}$
$\mathbf{\Omega}_n$
$\partial V_{s,\text{out}}$
$\mathbf{n}_{s,\text{out}}$

Figure 3.5: Depiction of a single slice showing relevant vectors and surfaces.

We begin by writing the energy-independent, steady-state, fixed source transport equation for angle n and slice s, assuming the total macroscopic cross section is constant within each cell, and hence also constant over each slice

$$\mathbf{\Omega}_n \cdot \nabla \psi_{n,s}\left(\mathbf{r}\right) + \sigma_{t,s}\psi_{n,s}\left(\mathbf{r}\right) = q_{n,s}\left(\mathbf{r}\right) \ . \tag{3.1}$$

The LDFE approximation is imposed by expanding the angular flux in the LDFE basis functions which are linear in space with local support

$$\psi_{n,s}(\mathbf{r}) = \sum_{i=c,x,y,z} b_i^s(\mathbf{r})\, \psi_{n,s}^i \ , \tag{3.2}$$

$$b_c^s(\mathbf{r}) = \begin{cases} 1 & \text{for } \mathbf{r} \in V_s \\ 0 & \text{otherwise} \end{cases} \ , \tag{3.3}$$

$$b_x^s(\mathbf{r}) = \begin{cases} (x - \overline{x}_s)/\Delta x_s & \text{for } \mathbf{r} \in V_s \\ 0 & \text{otherwise} \end{cases} \ , \tag{3.4}$$

$$b_y^s(\mathbf{r}) = \begin{cases} (y - \overline{y}_s)/\Delta y_s & \text{for } \mathbf{r} \in V_s \\ 0 & \text{otherwise} \end{cases} \ , \tag{3.5}$$

$$b_z^s(\mathbf{r}) = \begin{cases} (z - \overline{z}_s)/\Delta z_s & \text{for } \mathbf{r} \in V_s \\ 0 & \text{otherwise} \end{cases} \ , \tag{3.6}$$

where $\overline{x}_s$, $\overline{y}_s$, and $\overline{z}_s$ are the slice centroid coordinates, Δx_s, Δy_s, and Δz_s are the differences between the maximum and minimum value for each coordinate among the vertices of the slice, and $\psi_{n,s}^i$ are the constant coefficients of the LDFE basis function expansion. Next, we multiply equation 3.1 by each of the four basis functions, resulting in the four equations

$$b_i^s(\mathbf{r})\left\{\mathbf{\Omega}_n \cdot \nabla\psi_{n,s}(\mathbf{r}) + \sigma_{t,s}\psi_{n,s}(\mathbf{r})\right\} = b_i^s(\mathbf{r})\, q_{n,s}(\mathbf{r}) \quad \text{for} \quad i = c, x, y, z \ . \tag{3.7}$$

We then convert the first term into two terms by acknowledging that

$$\mathbf{\Omega}_n \cdot \nabla\left(b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\right) = \mathbf{\Omega}_n \cdot b_i^s(\mathbf{r})\,\nabla\psi_{n,s}(\mathbf{r}) + \mathbf{\Omega}_n \cdot \psi_{n,s}(\mathbf{r})\,\nabla b_i^s(\mathbf{r}) \ , \tag{3.8}$$

and equations 3.7 become

$$\begin{aligned}\boldsymbol{\Omega}_n \cdot \nabla\left(b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\right) - \boldsymbol{\Omega}_n \cdot \psi_{n,s}(\mathbf{r})\,\nabla b_i^s(\mathbf{r}) + \\ \sigma_{t,s}\, b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r}) = b_i^s(\mathbf{r})\,q_{n,s}(\mathbf{r}) \quad \text{for} \quad i = c, x, y, z\ .\end{aligned} \tag{3.9}$$

Integrating over the volume of the domain, which due to the local support of the basis functions is equivalent to integrating over the volume of the slice, gives

$$\begin{aligned}\boldsymbol{\Omega}_n \cdot \iiint\limits_{V_s} \nabla\left(b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\right)\, d^3r - \boldsymbol{\Omega}_n \cdot \iiint\limits_{V_s} \psi_{n,s}(\mathbf{r})\,\nabla b_i^s(\mathbf{r})\, d^3r + \\ \iiint\limits_{V_s} \sigma_{t,s}\, b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\, d^3r = \iiint\limits_{V_s} b_i^s(\mathbf{r})\,q_{n,s}(\mathbf{r})\, d^3r \quad \text{for} \quad i = c, x, y, z\ .\end{aligned} \tag{3.10}$$

We then convert the first volume integral into a surface integral

$$\begin{aligned}\boldsymbol{\Omega}_n \cdot \oiint\limits_{\partial V_s} \mathbf{n}(\mathbf{r})\, b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\, d^2r - \boldsymbol{\Omega}_n \cdot \iiint\limits_{V_s} \psi_{n,s}(\mathbf{r})\,\nabla b_i^s(\mathbf{r})\, d^3r + \\ \iiint\limits_{V_s} \sigma_{t,s}\, b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\, d^3r = \iiint\limits_{V_s} b_i^s(\mathbf{r})\,q_{n,s}(\mathbf{r})\, d^3r \quad \text{for} \quad i = c, x, y, z\ .\end{aligned} \tag{3.11}$$

Since $\boldsymbol{\Omega}_n \cdot \mathbf{n}(\mathbf{r})$ is zero on all faces except for $\partial V_{s,\text{in}}$ and $\partial V_{s,\text{out}}$, this becomes

$$\begin{aligned}(\boldsymbol{\Omega}_n \cdot \mathbf{n}_{s,\text{in}}) \iint\limits_{\partial V_{s,\text{in}}} b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\, d^2r + (\boldsymbol{\Omega}_n \cdot \mathbf{n}_{s,\text{out}}) \iint\limits_{\partial V_{s,\text{out}}} b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\, d^2r - \\ \boldsymbol{\Omega}_n \cdot \iiint\limits_{V_s} \psi_{n,s}(\mathbf{r})\,\nabla b_i^s(\mathbf{r})\, d^3r + \iiint\limits_{V_s} \sigma_{t,s}\, b_i^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\, d^3r = \\ \iiint\limits_{V_s} b_i^s(\mathbf{r})\,q_{n,s}(\mathbf{r})\, d^3r \quad \text{for} \quad i = c, x, y, z\ .\end{aligned} \tag{3.12}$$

We now insert the linear approximation for $\psi_{n,s}(\mathbf{r})$ into the volume integrals:

$$
\begin{aligned}
&(\boldsymbol{\Omega}_n \cdot \mathbf{n}_{s,\mathrm{in}}) \iint\limits_{\partial V_{s,\mathrm{in}}} b_i^s(\mathbf{r})\, \psi_{n,s}(\mathbf{r})\ d^2 r + (\boldsymbol{\Omega}_n \cdot \mathbf{n}_{s,\mathrm{out}}) \iint\limits_{\partial V_{s,\mathrm{out}}} b_i^s(\mathbf{r})\, \psi_{n,s}(\mathbf{r})\ d^2 r - \\
&\boldsymbol{\Omega}_n \cdot \iiint\limits_{V_s} \left(\sum_{j=c,x,y,z} b_j^s(\mathbf{r})\, \psi_{n,s}^j \right) \nabla b_i^s(\mathbf{r})\ d^3 r + \iiint\limits_{V_s} \sigma_{t,s}\, b_i^s(\mathbf{r}) \left(\sum_{j=c,x,y,z} b_j^s(\mathbf{r})\, \psi_{n,s}^j \right) d^3 r = \\
&\qquad \iiint\limits_{V_s} b_i^s(\mathbf{r})\, q_{n,s}(\mathbf{r})\ d^3 r \quad \text{for} \quad i = c,x,y,z\ . \qquad (3.13)
\end{aligned}
$$

We then re-arrange the summations and integrals to arrive at:

$$
\begin{aligned}
&(\boldsymbol{\Omega}_n \cdot \mathbf{n}_{s,\mathrm{in}}) \iint\limits_{\partial V_{s,\mathrm{in}}} b_i^s(\mathbf{r})\, \psi_{n,s}(\mathbf{r})\ d^2 r + (\boldsymbol{\Omega}_n \cdot \mathbf{n}_{s,\mathrm{out}}) \iint\limits_{\partial V_{s,\mathrm{out}}} b_i^s(\mathbf{r})\, \psi_{n,s}(\mathbf{r})\ d^2 r + \\
&\sum_{j=c,x,y,z} \iiint\limits_{V_s} \left(-\boldsymbol{\Omega}_n \cdot \nabla b_i^s(\mathbf{r})\, b_j^s(\mathbf{r})\, \psi_{n,s}^j + \sigma_{t,s}\, b_i^s(\mathbf{r})\, b_j^s(\mathbf{r})\, \psi_{n,s}^j \right) d^3 r = \\
&\qquad \iiint\limits_{V_s} b_i^s(\mathbf{r})\, q_{n,s}(\mathbf{r})\ d^3 r \quad \text{for} \quad i = c,x,y,z\ . \qquad (3.14)
\end{aligned}
$$

Finally, we divide the equation by the slice volume and multiply and divide each surface integral by the corresponding surface area to arrive at:

$$
\begin{aligned}
&\frac{A_{s,\mathrm{in}} (\boldsymbol{\Omega}_n \cdot \mathbf{n}_{s,\mathrm{in}})}{V_s} \left(\frac{1}{A_{s,\mathrm{in}}} \iint\limits_{\partial V_{s,\mathrm{in}}} b_i^s(\mathbf{r})\, \psi_{n,s}(\mathbf{r})\ d^2 r \right) + \\
&\qquad \frac{A_{s,\mathrm{out}} (\boldsymbol{\Omega}_n \cdot \mathbf{n}_{s,\mathrm{out}})}{V_s} \left(\frac{1}{A_{s,\mathrm{out}}} \iint\limits_{\partial V_{s,\mathrm{out}}} b_i^s(\mathbf{r})\, \psi_{n,s}(\mathbf{r})\ d^2 r \right) + \\
&\sum_{j=c,x,y,z} \frac{1}{V_s} \iiint\limits_{V_s} \left(-\boldsymbol{\Omega}_n \cdot \nabla b_i^s(\mathbf{r})\, b_j^s(\mathbf{r})\, \psi_{n,s}^j + \sigma_{t,s}\, b_i^s(\mathbf{r})\, b_j^s(\mathbf{r})\, \psi_{n,s}^j \right) d^3 r = \\
&\qquad \frac{1}{V_s} \iiint\limits_{V_s} b_i^s(\mathbf{r})\, q_{n,s}(\mathbf{r})\ d^3 r \quad \text{for} \quad i = c,x,y,z\ . \qquad (3.15)
\end{aligned}
$$

Next, we make the following definitions

$$\eta_{n,s,\mathrm{in}} = \frac{A_{s,\mathrm{in}}\left(\mathbf{\Omega}_n \cdot \mathbf{n}_{s,\mathrm{in}}\right)}{V_s} \ ,$$

$$\eta_{n,s,\mathrm{out}} = \frac{A_{s,\mathrm{out}}\left(\mathbf{\Omega}_n \cdot \mathbf{n}_{s,\mathrm{out}}\right)}{V_s} \ ,$$

$$\psi^i_{n,s,\mathrm{in}} = \frac{1}{A_{s,\mathrm{in}}} \iint\limits_{\partial V_{s,\mathrm{in}}} b^s_i\left(\mathbf{r}\right) \psi_{n,s}\left(\mathbf{r}\right)\, d^2r \ ,$$

$$\psi^i_{n,s,\mathrm{out}} = \frac{1}{A_{s,\mathrm{out}}} \iint\limits_{\partial V_{s,\mathrm{out}}} b^s_i\left(\mathbf{r}\right) \psi_{n,s}\left(\mathbf{r}\right)\, d^2r \ ,$$

$$q^i_{n,s} = \frac{1}{V_s} \iiint\limits_{V_s} b^s_i\left(\mathbf{r}\right) q_{n,s}\left(\mathbf{r}\right)\, d^3r \ ,$$

$$M^s_{ij} = \frac{1}{V_s} \iiint\limits_{V_s} b^s_i\left(\mathbf{r}\right) b^s_j\left(\mathbf{r}\right)\, d^3r \ .$$

With these definitions, our 4 equations can be written a bit more succinctly as

$$\eta_{n,s,\mathrm{in}}\psi^c_{n,s,\mathrm{in}} + \eta_{n,s,\mathrm{out}}\psi^c_{n,s,\mathrm{out}} + \sigma_{t,s}\psi^c_{n,s} = q^c_{n,s} \ , \tag{3.16}$$

$$\eta_{n,s,\mathrm{in}}\psi^x_{n,s,\mathrm{in}} + \eta_{n,s,\mathrm{out}}\psi^x_{n,s,\mathrm{out}} - \frac{1}{\Delta x_s}\Omega_{n,x}\psi^c_{n,s} + \sigma_{t,s}\left(M^s_{xx}\psi^x_{n,s} + M^s_{xy}\psi^y_{n,s} + M^s_{xz}\psi^z_{n,s}\right) = q^x_{n,s} \ , \tag{3.17}$$

$$\eta_{n,s,\mathrm{in}}\psi^y_{n,s,\mathrm{in}} + \eta_{n,s,\mathrm{out}}\psi^y_{n,s,\mathrm{out}} - \frac{1}{\Delta y_s}\Omega_{n,y}\psi^c_{n,s} + \sigma_{t,s}\left(M^s_{yx}\psi^x_{n,s} + M^s_{yy}\psi^y_{n,s} + M^s_{yz}\psi^z_{n,s}\right) = q^y_{n,s} \ , \tag{3.18}$$

$$\eta_{n,s,\mathrm{in}}\psi^{z}_{n,s,\mathrm{in}} + \eta_{n,s,\mathrm{out}}\psi^{z}_{n,s,\mathrm{out}} - \frac{1}{\Delta z_s}\Omega_{n,z}\psi^{c}_{n,s} + \sigma_{t,s}\left(M^{s}_{zx}\psi^{x}_{n,s} + M^{s}_{zy}\psi^{y}_{n,s} + M^{s}_{zz}\psi^{z}_{n,s}\right) = q^{z}_{n,s}\ . \quad (3.19)$$

Assuming that all variables obtained by integrating over the surface $\partial V_{s,\mathrm{in}}$ are known either from boundary conditions or values passed by upstream slices, there are eight unknowns in these four equations:

$$\psi^{c}_{n,s},\ \psi^{x}_{n,s},\ \psi^{y}_{n,s},\ \psi^{z}_{n,s},\ \psi^{c}_{n,s,\mathrm{out}},\ \psi^{x}_{n,s,\mathrm{out}},\ \psi^{y}_{n,s,\mathrm{out}},\ \psi^{z}_{n,s,\mathrm{out}}\ , \quad (3.20)$$

and thus, we need four more equations to solve for all eight unknowns. The remaining four equations come from the primary assumption that the angular flux is expanded in the LDFE basis functions. If we multiply equation 3.2 by the four basis functions and then integrate over the outlet surface and divide by the surface area, we arrive at four more equations

$$\psi^{i}_{n,s,\mathrm{out}} = \psi^{c}_{n,s}\left(\frac{1}{A_{s,\mathrm{out}}}\iint\limits_{\partial V_{s,\mathrm{out}}} b^{s}_{c}(\mathbf{r})\, b^{s}_{i}(\mathbf{r})\ d^{2}r\right) + \psi^{x}_{n,s}\left(\frac{1}{A_{s,\mathrm{out}}}\iint\limits_{\partial V_{s,\mathrm{out}}} b^{s}_{x}(\mathbf{r})\, b^{s}_{i}(\mathbf{r})\ d^{2}r\right) + \psi^{y}_{n,s}\left(\frac{1}{A_{s,\mathrm{out}}}\iint\limits_{\partial V_{s,\mathrm{out}}} b^{s}_{y}(\mathbf{r})\, b^{s}_{i}(\mathbf{r})\ d^{2}r\right) + \psi^{z}_{n,s}\left(\frac{1}{A_{s,\mathrm{out}}}\iint\limits_{\partial V_{s,\mathrm{out}}} b^{s}_{z}(\mathbf{r})\, b^{s}_{i}(\mathbf{r})\ d^{2}r\right) \quad \text{for} \quad i = c, x, y, z\ . \quad (3.21)$$

If we define

$$\alpha_{ij} = \frac{1}{A_{s,\mathrm{out}}}\iint\limits_{\partial V_{s,\mathrm{out}}} b^{s}_{i}(\mathbf{r})\, b^{s}_{j}(\mathbf{r})\ d^{2}r\ ,$$

we can write these four equations as

$$\psi^c_{n,s,\text{out}} = \alpha_{cc}\psi^c_{n,s} + \alpha_{cx}\psi^x_{n,s} + \alpha_{cy}\psi^y_{n,s} + \alpha_{cz}\psi^z_{n,s} \ , \tag{3.22}$$

$$\psi^x_{n,s,\text{out}} = \alpha_{xc}\psi^c_{n,s} + \alpha_{xx}\psi^x_{n,s} + \alpha_{xy}\psi^y_{n,s} + \alpha_{xz}\psi^z_{n,s} \ , \tag{3.23}$$

$$\psi^y_{n,s,\text{out}} = \alpha_{yc}\psi^c_{n,s} + \alpha_{yx}\psi^x_{n,s} + \alpha_{yy}\psi^y_{n,s} + \alpha_{yz}\psi^z_{n,s} \ , \tag{3.24}$$

$$\psi^z_{n,s,\text{out}} = \alpha_{zc}\psi^c_{n,s} + \alpha_{zx}\psi^x_{n,s} + \alpha_{zy}\psi^y_{n,s} + \alpha_{zz}\psi^z_{n,s} \ . \tag{3.25}$$

Plugging these expressions into equations 3.16 through 3.19 gives our final system of four equations and four unknowns, namely the coefficients of the LDFE basis function expansion, for each slice

$$\begin{aligned}(\eta_{n,s,\text{out}}\alpha_{cc} + \sigma_{t,s})\,\psi^c_{n,s} + (\eta_{n,s,\text{out}}\alpha_{cx})\,\psi^x_{n,s} + \\ (\eta_{n,s,\text{out}}\alpha_{cy})\,\psi^y_{n,s} + (\eta_{n,s,\text{out}}\alpha_{cz})\,\psi^z_{n,s} = q^c_{n,s} - \eta_{n,s,\text{in}}\psi^c_{n,s,\text{in}} \ ,\end{aligned} \tag{3.26}$$

$$\begin{aligned}(\eta_{n,s,\text{out}}\alpha_{xc} - \Omega_{n,x}/\Delta x_s)\,\psi^c_{n,s} + (\eta_{n,s,\text{out}}\alpha_{xx} + \sigma_{t,s}M^s_{xx})\,\psi^x_{n,s} + \\ \left(\eta_{n,s,\text{out}}\alpha_{xy} + \sigma_{t,s}M^s_{xy}\right)\psi^y_{n,s} + (\eta_{n,s,\text{out}}\alpha_{xz} + \sigma_{t,s}M^s_{xz})\,\psi^z_{n,s} = \\ q^x_{n,s} - \eta_{n,s,\text{in}}\psi^x_{n,s,\text{in}} \ ,\end{aligned} \tag{3.27}$$

$$\begin{aligned}(\eta_{n,s,\text{out}}\alpha_{yc} - \Omega_{n,y}/\Delta y_s)\,\psi^c_{n,s} + \left(\eta_{n,s,\text{out}}\alpha_{yx} + \sigma_{t,s}M^s_{yx}\right)\psi^x_{n,s} + \\ \left(\eta_{n,s,\text{out}}\alpha_{yy} + \sigma_{t,s}M^s_{yy}\right)\psi^y_{n,s} + \left(\eta_{n,s,\text{out}}\alpha_{yz} + \sigma_{t,s}M^s_{yz}\right)\psi^z_{n,s} = \\ q^y_{n,s} - \eta_{n,s,\text{in}}\psi^y_{n,s,\text{in}} \ ,\end{aligned} \tag{3.28}$$

$$\left(\eta_{n,s,\mathrm{out}}\alpha_{zc} - \Omega_{n,z}/\Delta z_s\right)\psi^c_{n,s} + \left(\eta_{n,s,\mathrm{out}}\alpha_{zx} + \sigma_{t,s}M^s_{zx}\right)\psi^x_{n,s} + \left(\eta_{n,s,\mathrm{out}}\alpha_{zy} + \sigma_{t,s}M^s_{zy}\right)\psi^y_{n,s} + \left(\eta_{n,s,\mathrm{out}}\alpha_{zz} + \sigma_{t,s}M^s_{zz}\right)\psi^z_{n,s} = q^z_{n,s} - \eta_{n,s,\mathrm{in}}\psi^z_{n,s,\mathrm{in}} \,. \quad (3.29)$$

While these equations are sufficient to determine the spatial dependence of the angular flux within slice s, eventually what we want is the spatial dependence of the angular flux within the cell c from which slice s was formed. To do this, we first define the LDFE basis functions for cell c, and expand the angular flux within the cell in these basis functions as well

$$\psi_{n,c}(\mathbf{r}) = \sum_{i=c,x,y,z} b^c_i(\mathbf{r})\,\psi^i_{n,c} \,, \quad (3.30)$$

$$b^c_c(\mathbf{r}) = \begin{cases} 1 & \text{for } \mathbf{r} \in V_c \\ 0 & \text{otherwise} \end{cases} \,, \quad (3.31)$$

$$b^c_x(\mathbf{r}) = \begin{cases} \left(x - \overline{x}_c\right)/\Delta x_c & \text{for } \mathbf{r} \in V_c \\ 0 & \text{otherwise} \end{cases} \,, \quad (3.32)$$

$$b^c_y(\mathbf{r}) = \begin{cases} \left(y - \overline{y}_c\right)/\Delta y_c & \text{for } \mathbf{r} \in V_c \\ 0 & \text{otherwise} \end{cases} \,, \quad (3.33)$$

$$b^c_z(\mathbf{r}) = \begin{cases} \left(z - \overline{z}_c\right)/\Delta z_c & \text{for } \mathbf{r} \in V_c \\ 0 & \text{otherwise} \end{cases} \,, \quad (3.34)$$

where $\overline{x}_c$, $\overline{y}_c$, and $\overline{z}_c$ are the cell centroid coordinates, Δx_c, Δy_c, and Δz_c are the differences between the maximum and minimum value for each coordinate among the vertices of the cell, and $\psi_{n,c}^{i}$ are the constant coefficients of the LDFE basis function expansion. In order to find the expansion coefficients, we multiply equation 3.30 by each of the cell basis functions and integrate over, and divide by, the volume of the cell

$$\frac{1}{V_c}\iiint\limits_{V_c} b_i^c(\mathbf{r})\,\psi_{n,c}(\mathbf{r})\ d^3r = \psi_{n,c}^{c}\left(\frac{1}{V_c}\iiint\limits_{V_c} b_i^c(\mathbf{r})\,b_c^c(\mathbf{r})\ d^3r\right) + \psi_{n,c}^{x}\left(\frac{1}{V_c}\iiint\limits_{V_c} b_i^c(\mathbf{r})\,b_x^c(\mathbf{r})\ d^3r\right) + \psi_{n,c}^{y}\left(\frac{1}{V_c}\iiint\limits_{V_c} b_i^c(\mathbf{r})\,b_y^c(\mathbf{r})\ d^3r\right) + \psi_{n,c}^{z}\left(\frac{1}{V_c}\iiint\limits_{V_c} b_i^c(\mathbf{r})\,b_z^c(\mathbf{r})\ d^3r\right) \quad \text{for} \quad i = c, x, y, z\ . \tag{3.35}$$

Again, defining

$$M_{ij}^{c} = \frac{1}{V_c}\iiint\limits_{V_c} b_i^c(\mathbf{r})\,b_j^c(\mathbf{r})\ d^3r\ ,$$

we can write this as

$$\frac{1}{V_c}\iiint\limits_{V_c} b_i^c(\mathbf{r})\,\psi_{n,c}(\mathbf{r})\ d^3r = M_{ic}^{c}\psi_{n,c}^{c} + M_{ix}^{c}\psi_{n,c}^{x} + M_{iy}^{c}\psi_{n,c}^{y} + M_{iz}^{c}\psi_{n,c}^{z} \quad \text{for} \quad i = c, x, y, z\ . \tag{3.36}$$

This too is a system of four equations and four unknowns, namely the LDFE expansion coefficients for the angular flux in the cell, however it is not immediately obvious how to obtain the left hand sides in order to solve this system for each cell in the mesh. It turns out that as the angular flux in each slice is solved for, these integrals on the left hand side can be accumulated. To illustrate this, consider first the $i = c$ case

$$\frac{1}{V_c}\iiint\limits_{V_c} b_c^c(\mathbf{r})\,\psi_{n,c}(\mathbf{r})\;d^3r = \frac{1}{V_c}\iiint\limits_{V_c}\psi_{n,c}(\mathbf{r})\;d^3r = \frac{1}{V_c}\sum_s\iiint\limits_{V_s}\psi_{n,s}(\mathbf{r})\;d^3r\;, \tag{3.37}$$

where the sum over s indicates a sum over all slices contained in cell c. Fortunately, this case is relatively straightforward, since the integrals of the non-constant basis functions over the slice volume equate to zero, and the integral inside the summation reduces to $V_s\psi_{n,s}^c$. Thus

$$\frac{1}{V_c}\iiint\limits_{V_c} b_c^c(\mathbf{r})\,\psi_{n,c}(\mathbf{r})\;d^3r = \frac{1}{V_c}\sum_s V_s\psi_{n,s}^c\;. \tag{3.38}$$

For $i = x, y$, or z, things are only slightly more complicated. Consider for example the $i = x$ case

$$\frac{1}{V_c}\iiint\limits_{V_c} b_x^c(\mathbf{r})\,\psi_{n,c}(\mathbf{r})\;d^3r = \frac{1}{V_c}\sum_s\iiint\limits_{V_s} b_x^c(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\;d^3r\;. \tag{3.39}$$

If the basis function inside the integral inside the summation were the slice basis function, this case would be obvious as well. Unfortunately it is not, and we must express the cell basis function in terms of the slice basis function

$$\begin{aligned} b_x^c(\mathbf{r}) = \frac{(x-\overline{x}_c)}{\Delta x_c} = \frac{(x-\overline{x}_s+\overline{x}_s-\overline{x}_c)}{\Delta x_c\frac{\Delta x_s}{\Delta x_s}} = \\ \frac{\Delta x_s}{\Delta x_c}\left(\frac{(x-\overline{x}_s)}{\Delta x_s}+\frac{(\overline{x}_s-\overline{x}_c)}{\Delta x_s}\right) = \frac{\Delta x_s}{\Delta x_c}\left(b_x^s(\mathbf{r})+\frac{(\overline{x}_s-\overline{x}_c)}{\Delta x_s}\right)\;. \end{aligned} \tag{3.40}$$

Substituting this expression for $b_x^c(\mathbf{r})$ in the right hand side of equation 3.39 gives

$$\frac{1}{V_c}\iiint\limits_{V_c} b_x^c(\mathbf{r})\,\psi_{n,c}(\mathbf{r})\;d^3r =$$

$$\frac{1}{V_c}\sum_s \frac{\Delta x_s}{\Delta x_c}\left(\iiint\limits_{V_s} b_x^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\;d^3r + \frac{(\overline{x}_s-\overline{x}_c)}{\Delta x_s}\iiint\limits_{V_s}\psi_{n,s}(\mathbf{r})\;d^3r\right) =$$

$$\frac{1}{V_c}\sum_s \frac{\Delta x_s}{\Delta x_c}\left(\iiint\limits_{V_s} b_x^s(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\;d^3r + \frac{(\overline{x}_s-\overline{x}_c)}{\Delta x_s}V_s\psi_{n,s}^c\right)\;. \quad (3.41)$$

Performing the same manipulations for the $i = y$ and $i = z$ cases result in similar summations. While the quantities in parentheses still look quite complicated, it should be noted that upon expansion of $\psi_{n,s}(\mathbf{r})$ inside the last remaining integral, this reduces to a linear combination of the slice LDFE basis function expansion coefficients, with the multipliers being the M_{ij}^s already computed. Therefore, this sum can be accumulated into as the angular flux in each slice is solved for. After the angular flux has been computed in all slices, and the left hand sides of equations 3.36 have all been accumulated, equations 3.36 can be inverted independently for each cell in the mesh.

As a side note, for steady state problems, storing the entire angular flux solution is unnecessary. In this case, it is sufficient to store the scalar flux moments

$$\phi_l^m(\mathbf{r}) = \iint\limits_{4\pi} Y_l^m(\mathbf{\Omega})\,\psi(\mathbf{r},\mathbf{\Omega})\;d\Omega\;. \quad (3.42)$$

where $Y_l^m(\mathbf{\Omega})$ are the tesseral spherical harmonics. These moments are expanded in the same LDFE basis functions as the angular flux, and the integral over all angles is performed by quadrature integration using the S_N quadrature set

$$(\phi_l^m)_c^c = \sum_n \omega_n Y_l^m(\mathbf{\Omega}_n)\, \psi_{n,c}^c \ ,$$

$$(\phi_l^m)_c^x = \sum_n \omega_n Y_l^m(\mathbf{\Omega}_n)\, \psi_{n,c}^x \ ,$$

$$(\phi_l^m)_c^y = \sum_n \omega_n Y_l^m(\mathbf{\Omega}_n)\, \psi_{n,c}^y \ ,$$

$$(\phi_l^m)_c^z = \sum_n \omega_n Y_l^m(\mathbf{\Omega}_n)\, \psi_{n,c}^z \ .$$

where ω_n are the S_N quadrature weights and $\mathbf{\Omega}_n$ are the S_N are the quadrature nodes. The number of moments necessary to store is dependent upon what order the scattering source expansion is terminated at. It can therefore be very economical to solve

$$\sum_n \omega_n Y_l^m(\mathbf{\Omega}_n) \frac{1}{V_c} \iiint\limits_{V_c} b_i^c(\mathbf{r})\, \psi_{n,c}(\mathbf{r})\ d^3r =$$
$$M_{ic}^c (\phi_l^m)_c^c + M_{ix}^c (\phi_l^m)_c^x + M_{iy}^c (\phi_l^m)_c^y + M_{iz}^c (\phi_l^m)_c^z \quad \text{for} \quad i = c, x, y, z \ , \tag{3.43}$$

instead of equations 3.36 for each cell. The sums are accumulated in much the same way, however fewer unknowns per cell must be solved for and stored than if the entire angular flux were required.

After the angular flux in a slice has been determined, this information must be passed to the downstream sub-slices. As alluded to earlier in this section, we are using the upwind approximation, which is to say that the angular flux on the sub-slice incoming face is determined by the expression for the angular flux in the slice directly upstream of it. For instance, consider Figure 3.6 which shows a slice in red, and the four sub-slices formed from its outlet face in green, yellow, blue and purple. Each of these four sub-slices will use the LDFE expansion of the angular flux in the

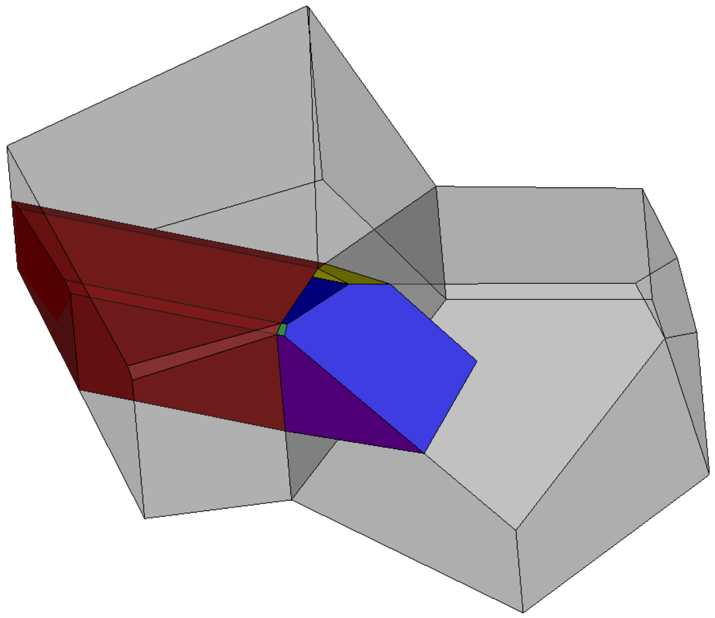


Figure 3.6: Three dimensional illustration of a slice, shown in red, and the four sub-slices formed from its outlet face shown in green, yellow, blue, and purple.

upstream slice in place of $\psi_{n,ss}(\mathbf{r})$ to evaluate the weighted integrals of the angular flux on their incoming faces

$$\psi^{i}_{n,ss,\text{in}} = \frac{1}{A_{s,\text{in}}} \iint\limits_{\partial V_{ss,\text{in}}} b^{ss}_{i}(\mathbf{r})\, \psi_{n,ss}(\mathbf{r})\, d^2 r \quad \text{for} \quad i = c, x, y, z\ , \tag{3.44}$$

where $b^{ss}_{i}(\mathbf{r})$ are the LDFE basis functions for the sub-slice, indexed ss, which are defined similarly to those already given for the slice and the cell. In this way, all incoming fluxes are communicated via the sub-slices, and not the cell faces as in the traditional SBA. We further note that each slice, unless its inlet face is on the incoming portion of the domain boundary, is composed of sub-slices that are non-overlapping and fill the volume of the slice. A slice can be considered ready to be solved when all of its contained sub-slices have received information from their upstream slice and computed their $\psi^{i}_{n,ss,\text{in}}$.

Once the sub-slices contained within a slice have received this information, the incoming fluxes of these sub-slices can be appropriately coalesced to give the remaining information on the right hand side of equations 3.26 through 3.29, namely $\psi^i_{n,s,\mathrm{in}}$. This is done in a similar manner as the LDFE coefficients for the angular flux within slices were coalesced to give the LDFE coefficients of the angular flux throughout the parent cell. To illustrate this, consider first the $i = c$ case

$$\psi^i_{n,s,\mathrm{in}} = \frac{1}{A_{s,\mathrm{in}}} \iint\limits_{\partial V_{s,\mathrm{in}}} b^s_c(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\ d^2r = \frac{1}{A_{s,\mathrm{in}}} \iint\limits_{\partial V_{s,\mathrm{in}}} \psi_{n,s}(\mathbf{r})\ d^2r = \frac{1}{A_{s,\mathrm{in}}} \sum_{ss} \iint\limits_{\partial V_{ss,\mathrm{in}}} \psi_{n,ss}(\mathbf{r})\ d^2r = \frac{1}{A_{s,\mathrm{in}}} \sum_{ss} A_{ss,\mathrm{in}} \psi^c_{n,ss,\mathrm{in}}\ , \tag{3.45}$$

where the sum over ss indicates a sum over all sub-slices contained in slice s. Again, while this case was relatively straightforward, the $i = x, y$, and z cases are only slightly more complicated. Consider for example the $i = x$ case

$$\psi^x_{n,s,\mathrm{in}} = \frac{1}{A_{s,\mathrm{in}}} \iint\limits_{\partial V_{s,\mathrm{in}}} b^s_x(\mathbf{r})\,\psi_{n,s}(\mathbf{r})\ d^2r = \frac{1}{A_{s,\mathrm{in}}} \sum_{ss} \iint\limits_{\partial V_{ss,\mathrm{in}}} b^s_x(\mathbf{r})\,\psi_{n,ss}(\mathbf{r})\ d^2r \tag{3.46}$$

Again, if the basis function inside the integral inside the summation were the sub-slice basis function, this case would be obvious as well. Unfortunately it is not, and we must express the slice basis function in terms of the sub-slice basis function

$$b^s_x(\mathbf{r}) = \frac{(x - \overline{x}_s)}{\Delta x_s} = \frac{(x - \overline{x}_{ss} + \overline{x}_{ss} - \overline{x}_s)}{\Delta x_s \frac{\Delta x_{ss}}{\Delta x_{ss}}} = \frac{\Delta x_{ss}}{\Delta x_s}\left(\frac{(x - \overline{x}_{ss})}{\Delta x_{ss}} + \frac{(\overline{x}_{ss} - \overline{x}_s)}{\Delta x_{ss}}\right) = \frac{\Delta x_{ss}}{\Delta x_s}\left(b^{ss}_x(\mathbf{r}) + \frac{(\overline{x}_{ss} - \overline{x}_s)}{\Delta x_{ss}}\right)\ . \tag{3.47}$$

Substituting this expression for $b^s_x(\mathbf{r})$ in the right hand side of equation 3.46 gives

$$\psi^x_{n,s,\text{in}} =$$

$$\frac{1}{A_{s,\text{in}}} \sum_{ss} \frac{\Delta x_{ss}}{\Delta x_s} \left(\iint\limits_{\partial V_{ss,\text{in}}} b^{ss}_x(\mathbf{r})\, \psi_{n,ss}(\mathbf{r})\ d^2 r + \frac{(\overline{x}_{ss} - \overline{x}_s)}{\Delta x_{ss}} \iint\limits_{\partial V_{ss,\text{in}}} \psi_{n,ss}(\mathbf{r})\ d^2 r \right) =$$

$$\frac{1}{A_{s,\text{in}}} \sum_{ss} \frac{\Delta x_{ss}}{\Delta x_s} \left(A_{ss,\text{in}} \psi^x_{n,ss,\text{in}} + \frac{(\overline{x}_{ss} - \overline{x}_s)}{\Delta x_{ss}} A_{ss,\text{in}} \psi^c_{n,ss,\text{in}} \right) . \quad (3.48)$$

Performing the same manipulations for the $i = y$ and $i = z$ cases result in similar summations. In this way, sub-slice incoming flux information can be combined to give the incoming flux information for the parent slice.

3.3 Local Sweep Description

In the previous chapter, the parallel transport sweep was discussed in the context of a domain decomposed mesh in which each node of a super-computer or cluster, stored and was responsible for computing, the solution on the cells contained in that node's subset of the mesh. The process of solving for the solution on the subset was not mentioned, however this local solve can also be carried out via sweeping. This local sweep is equivalent to solving a lower triangular matrix equation whose solution is the angular flux or scalar flux moments on each cell of the subset of the domain. In this section, we provide an algorithm for performing this local sweep in the context of the extended SBA using the LDFE spatial discretization scheme presented above. The full algorithm is described in detail in Algorithm 3.1. The sweep can be viewed as a loop over stages, where a sufficiently high stage count is chosen to ensure completion of the sweep, but to avoid infinite loops in case of unforeseen errors.

Algorithm 3.1: Local transport sweep for angle $\mathbf{\Omega}_n$.

```
1: for $k = 0$ to $N_{\text{max stages}} - 1$ do
2:     if $k = 0$ then
3:         for $s = 0$ to $N_{\text{slices}} - 1$ do
4:             $f$ = inlet face index for slice $s$
5:             if face $f$ is on the incoming boundary then
6:                 get $\psi^c_{n,s,\text{in}}, \psi^x_{n,s,\text{in}}, \psi^y_{n,s,\text{in}}$, and $\psi^z_{n,s,\text{in}}$ from the boundary conditions
7:                 add slice $s$ to the Ready queue
8:                 mark slice $s$ as done
9:             end if
10:        end for
11:    end if
12:    for $l = 0$ to $N_{\text{Ready}} - 1$ do
13:        get slice index $s$ from the Ready queue
14:        solve equations 3.26 through 3.29 to get $\psi^c_{n,s}, \psi^x_{n,s}, \psi^y_{n,s}$, and $\psi^z_{n,s}$
15:        atomic: contribute to LHS of equations 3.36 or 3.43
16:    end for
17:    for $l = 0$ to $N_{\text{Ready}} - 1$ do
18:        get slice index $s$ from the Ready queue
19:        for $i = 0$ to $N_{\text{sub-slices downstream of slice } s} - 1$ do
20:            get sub-slice index $ss$
21:            add sub-slice $ss$ to the Pending queue
22:            get slice index $s^*$ that sub-slice $ss$ is contained in
23:            increment the number of pending sub-slices contained in slice $s^*$
24:            get $\psi^c_{n,ss,\text{in}}, \psi^x_{n,ss,\text{in}}, \psi^y_{n,ss,\text{in}}$, and $\psi^z_{n,ss,\text{in}}$ from equation 3.44
25:        end for
26:    end for
27:    empty the Ready queue
28:    for $s = 0$ to $N_{\text{slices}} - 1$ do
29:        if the number of pending sub-slices contained in slice $s$ is equal to
30:           the number of sub-slices contained in slice $s$ and slice $s$ is not
31:           done then
32:            add slice $s$ to the Ready queue
33:            mark slice $s$ as done
34:            mark all sub-slices contained in slice $s$ as done
35:            get $\psi^c_{n,s,\text{in}}, \psi^x_{n,s,\text{in}}, \psi^y_{n,s,\text{in}}$, and $\psi^z_{n,s,\text{in}}$ from equations 3.45 and
36:                                                  3.48 (for $i = x, y$, and $z$)
37:        end if
38:    end for
```

```
39:     remove all sub-slices marked done from the Pending queue
40:     **if** $N_{\text{Ready}} = 0$ **then**
41:         break from the stage loop
42:     **end if**
43: **end for**
```

To begin the sweep, when the stage index is equal to zero, a loop over the slices determines which slices have their incoming faces on the domain boundary. For these slices, the incoming flux integrals are determined from the boundary conditions of the problem. Each of these slices is marked done, and their indices are added to a "Ready" queue, signifying that they are ready to be solved. This is only done for the first stage, as the Ready queue is emptied and refilled at the end of the stage loop.

With a filled Ready queue, each slice in the queue can be solved for independently, using equations 3.26 through 3.29. This introduces an opportunity for parallelism among the shared memory cores of the node on which the sweep is taking place. After the solution is determined on each slice, the contributions to the left hand sides of either equations 3.36 or 3.43 (depending upon whether the angular flux is required, or whether the scalar flux moments will suffice) can be made. Care must be taken to avoid race conditions in the case of slices of the same cell attempting to add their contributions to the left hand sides simultaneously.

After all slices in the Ready queue have been solved, the slices in the Ready queue are looped over again. For each slice in the queue, we loop over the sub-slices downstream of the slice, and add each one to a "Pending" queue, signifying that they have their incoming flux information. The number of pending sub-slices in the parent slice is incremented, and the incoming flux integrals on each sub-slice are computed using equation 3.44. Once this is completed, we empty the Ready queue, and prepare to fill it back up for the next stage.

To refill the Ready queue, we loop over the slices and check if the number of pending sub-slices within the slice equals the total number of sub-slices within the slice, while also checking to make sure the slice has not been marked done. If these conditions are satisfied, then it is ready to be solved, and can be placed in the Ready queue. We place it in the queue and mark it and all of its contained sub-slices as done, and then use equations 3.45 and 3.48 for $i = x, y$, and z in order to coalesce the incoming flux information for the contained sub-slices into the incoming flux information for the slice.

Finally, we loop through the Pending queue and remove all sub-slices that have been marked done. At the very end of the loop, we check the size of the Ready queue. If the size of the Ready queue is zero, this means that all slices in the domain have been solved, and the local sweep for angle $\mathbf{\Omega}_n$ is complete. Alternatively, one could also add a loop to check that all slices and sub-slices have been marked as done if one is overly suspicious that the entire domain has been swept.

4. PARALLELIZATION

Where the previous chapter emphasized the theory of the extended slice balance approach (ESBA), this chapter will focus primarily upon its implementation and parallelization strategies. As alluded to in the previous chapter, the extension of the Slice Balance Approach (SBA) to include sub-slices allows for the division of the spatial domain into regions separated by planes in which the solution in each region is independent of the solution in all other such regions. This introduces more concurrency which may be taken advantage of by parallel architectures such as super-computers and clusters. Two parallelization options that are made possible by this development will be discussed in this chapter. In addition, this chapter will discuss the incorporation of graphics processing units (GPUs) into the SBA and ESBA to make the linear discontinuous finite element (LDFE) spatial discretization more viable.

4.1 Parallelization Option 1

The first parallelization option to be presented is quite similar to the traditional transport sweep, in that each node is responsible for computing and storing the transport solution on the subset of the mesh assigned to it, and fluxes are communicated on the faces shared by neighboring nodes. As in traditional sweeps, this option does require planar inter-node domain boundaries, and thus a method for decomposing and load-balancing arbitrary polyhedral meshes into brick shaped domains has been developed by building on previous work by Ghaddar, who developed a method for load balancing extruded triangular meshes into brick shaped regions.[22] The decomposition algorithm developed here is essentially a recursive analog to a one dimensional version of Ghaddar's algorithm, and is presented in Appendix A.

4.1.1 Description

To illustrate parallelization option 1, consider the two dimensional mesh shown in Figure 4.1a, where different colors represent different materials, each meshed with a different resolution. After applying the load-balancing brick decomposition algorithm to this simple mesh, the result is the mesh shown in Figure 4.1b, where each color represents a different subset of the mesh assigned to a different node of a supercomputer or cluster. We note that "cut planes" have been introduced and that each cut plane cuts some cells in two, adding to the cell count in the mesh.

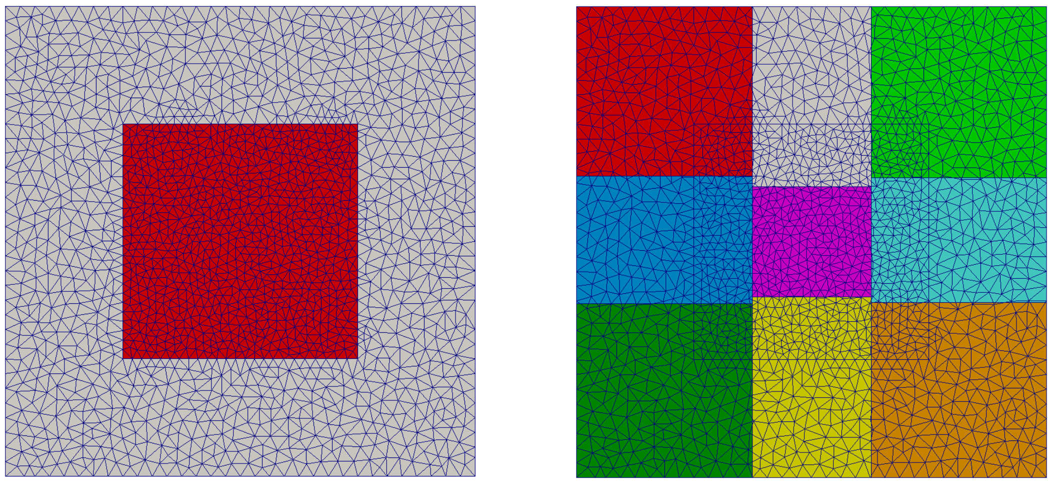

(a) Original two material triangular mesh. (b) After decomposition.

Figure 4.1: Two dimensional triangular mesh to illustrate parallelization option 1.

Next, we define a patch as the set of boundary faces (or edges in this two dimensional example) on each node which either lie on a planar portion of the problem boundary, or are shared by a unique pair of nodes. Next, we define a task as an inlet patch-angle pair, where each inlet patch is projected through the mesh on each node in the direction of $\mathbf{\Omega}_n$. This is illustrated in Figure 4.2.

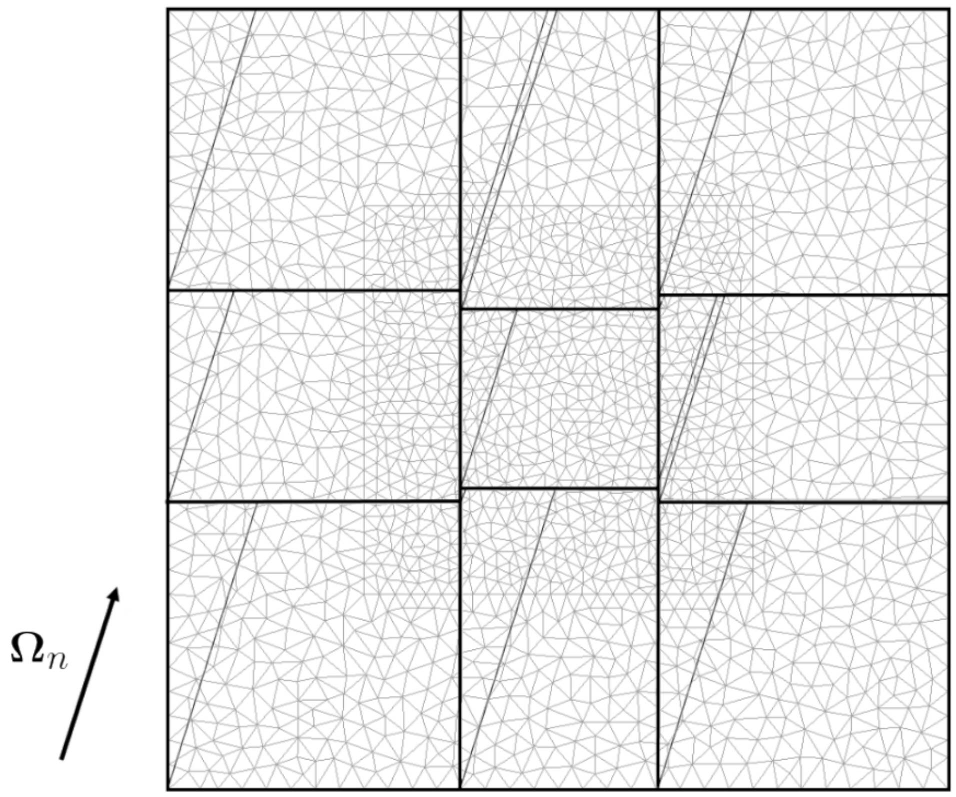


Figure 4.2: Illustration of tasks for angle $\mathbf{\Omega}_n$.

The sweep ordering is depicted in Figure 4.3. To begin the sweep, we note that there are six tasks colored in red that can begin solving in the first stage, since their inlet patches lie on the problem boundary. As each task obtains the transport solution in its region, it stores the fluxes on the outgoing patch faces, and once the fluxes on all faces on an outgoing patch are computed, the node communicates this patch's worth of boundary fluxes to the node downstream of the outgoing patch. In the second stage, the four tasks colored in blue have received their incoming boundary information, and can begin computing in their regions. This continues through the green, yellow, orange, purple, and finally pink stages.

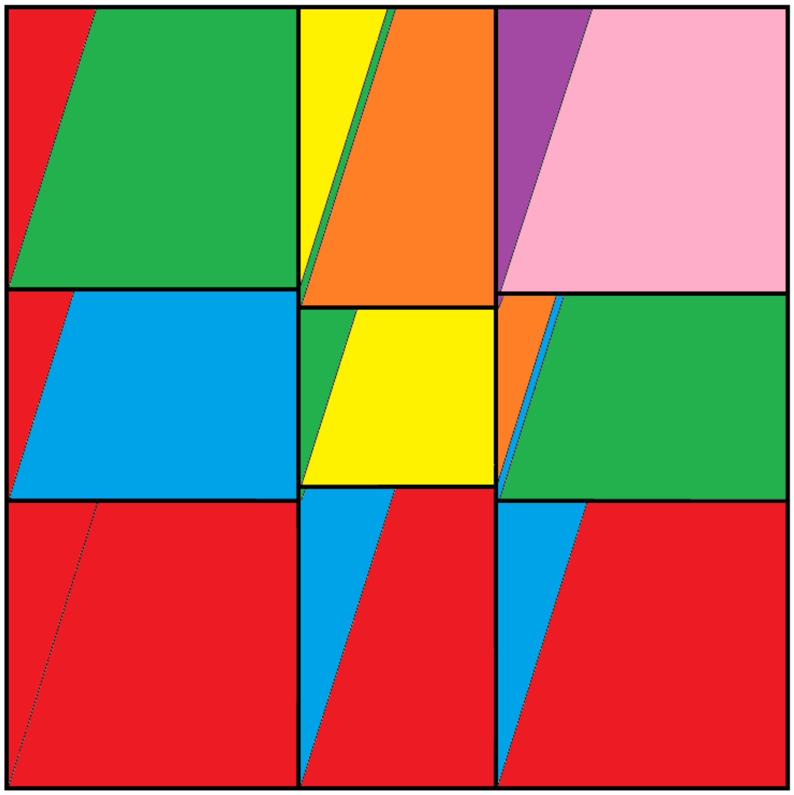


Figure 4.3: Tasks colored by order in the sweep for angle $\mathbf{\Omega}_n$.

The end result is a reduction of idle time on the front end of the sweep due to the additional concurrency that the concept of the sub-slice provides. This should theoretically increase the ceiling for the parallel efficiency of the transport sweep, while also increasing the accuracy of the solution over the traditional CBA. To determine the order in which each node performs its tasks, each task is assigned a weight based on the number of slices on downstream nodes that are dependent on this task finishing. This heuristic task ordering aims to get information through the mesh as quickly as possible. Also note, that this provides a natural way to use multi-core nodes, since the tasks are completely independent and can be performed simultaneously through shared memory parallelism with very little overhead.

4.1.2 Algorithm

The global sweep algorithm for parallelization option 1 is shown in Algorithm 4.1, which is to be executed on each node. The entire algorithm is enclosed in a threaded region signifying shared memory parallelism by the cores on each node. Within this threaded region, each thread performs a loop over the tasks on this node, which have already been ordered heuristically as mentioned in the last section. Thus, task 0 will have the largest weight, corresponding to the number of slices on all other nodes that rely on this task's execution in order to receive incoming boundary information, whether directly or indirectly.

At the beginning of the task loop, a query is made as to whether or not the current task has already been executed. If so, a further check is performed to see if this is the last task in the task list. If it is indeed the last task in the list, the loop index is reset to -1 so that on the next iteration it will restart from zero, assuming the loop index is incremented at the end of each iteration. A final check is then made to determine whether all tasks have been executed, and if so, the thread breaks from the task loop. If the task has been executed, and it is not the last task in the task list, the rest of the task loop is bypassed to move on to the next task in the list.

The next step is to declare a map of slice flux communication structures. Such a structure should contain the information defining a slice, namely the inlet and outlet face indices on the node owning its parent cell, the energy group index to which the flux belongs, and the incoming angular flux moments $\psi^c_{n,s,\text{in}}$, $\psi^x_{n,s,\text{in}}$, $\psi^y_{n,s,\text{in}}$, and $\psi^z_{n,s,\text{in}}$. Entries in this map are accessed via a tuple of the inlet face, outlet face, and energy group indices. A boolean used to signify whether this task is able to be executed is initially set to false, and the inlet patch and angle indices for this task are stored.

With the task information gathered, and the incoming flux map declared, the

Algorithm 4.1: Global transport sweep for parallelization option 1.

```
begin threaded region
    for $i = 0$ to $N_{\text{tasks}} - 1$ do
        if task $i$ is done then
            if $i = N_{\text{tasks}} - 1$ then
                $i = -1$
                if all tasks are done then
                    break from the task loop
                end if
            end if
            continue to next task
        end if
        incoming = map<tuple<int, int, int>, commStruct>
        gotTask = false
        $p$ = incoming patch index for task $i$
        $m$ = angle index for task $i$
        begin critical region 1
            if task $i$ is not done then
                if patch $p$ is on an inter-node domain boundary then
                    probe for message with label $m$ from node sharing patch $p$
                    if a message is waiting to be received then
                        receive a vector of commStructs called boundFluxes
                        for $j = 0$ to $N_{\text{boundFluxes}} - 1$ do
                            inF = boundFluxes[$j$].inF
                            outF = boundFluxes[$j$].outF
                            g = boundFluxes[$j$].g
                            incoming[(inF, outF, g)] = boundFluxes[$j$]
                        end for
                        mark task $i$ as done
                        gotTask = true
                        $i = -1$
                    end if
                else if patch $p$ is on the problem boundary then
                    mark task $i$ as done
                    gotTask = true
                    $i = -1$
                end if
            end if
        end critical region 1
        if gotTask = true then
            perform local sweep for task $i$ using Algorithm 3.1
            while sweeping, collect and count commStructs of slices in ghost cells
```

```
                        into a 2-D vector of dimension $N_{\text{patches}} \times N_{\text{slices per outlet patch}}$
                        named tempOutgoing
                **begin critical region 2**
                    **for** $j = 0$ to $N_{\text{patches}} - 1$ **do**
                        **if** outgoing[$m$][$j$].size = 0 **then**
                            **for** $k = 0$ to tempOutgoing[$j$].size $-1$ **do**
                                append tempOutgoing[$j$][$k$] to outgoing[$m$][$j$]
                            **end for**
                        **else**
                            **for** $k = 0$ to tempOutgoing[$j$].size $-1$ **do**
                                inF = tempOutgoing[$j$][$k$].inF
                                outF = tempOutgoing[$j$][$k$].outF
                                g = tempOutgoing[$j$][$k$].g
                                check if outgoing[$m$][$j$] already contains a commStruct
                                        defined by the tuple (inF, outF, g)
                                **if** so **then**
                                    add inlet fluxes for tempOutgoing[$j$][$k$] to the
                                            matching entry in outgoing[$m$][$j$]
                                **else**
                                    append tempOutgoing[$j$][$k$] to outgoing[$m$][$j$]
                                **end if**
                            **end for**
                        **end if**
                    **end for**
                    **for** $j = 0$ to $N_{\text{patches}} - 1$ **do**
                        patchSliceCount[$m$][$j$] += slicesPerOutPatch[$j$]
                        **if** patchSliceCount[$m$][$j$] = $N_{\text{slices}}$[$m$][$j$] **and**
                                outPatchDone[$m$][$j$] = false **then**
                            outPatchDone[$m$][$j$] = true
                            **send** outgoing[$m$][$j$] to node sharing patch $p$ with label $m$
                        **end if**
                    **end for**
                **end critical region 2**
            **end if**
            **if** $i = N_{\text{tasks}} - 1$ **then**
                $i = -1$
                **if** all tasks are done **then**
                    **break** from the task loop
                **end if**
            **end if**
        **end for**
    **end threaded region**
```

algorithm then enters the first critical region. A critical region in this case simply means that only one thread can enter the region at a time. Once inside this critical region, a thread will once again check if the task has been executed, because it is possible that since this thread has re-entered the task loop, the current task may have been allocated to another thread. It is then determined whether the incoming patch is on an inter-node domain boundary, or whether it lies on the problem boundary. If it is on an inter-node domain boundary, the thread probes for a message from the node on the other side of the patch with a label corresponding to the task angle. If a message is indeed waiting, a vector of flux communication structures is received and placed into the incoming flux map. If the inlet patch is on the problem boundary, the incoming flux map will be populated via the boundary conditions as part of the local sweep to be performed in the next step. In either case, the current task is marked done, the boolean signifying whether a task ready for execution was found is set to true, and the task loop index is reset to -1 so that on the next iteration it will restart from zero, all while still inside the first critical region.

If the thread found a task that is ready for execution, it then performs a local sweep of the slices in this task immediately after exiting the first critical region. What is communicated between nodes is actually not cell face fluxes, but fluxes on incoming faces of slices residing on the node sharing the patch. It is therefore necessary that each node store the geometric information of the cells on the other side of the patch faces, and these cells are referred to as "ghost" cells. While performing the local sweep, the thread will keep count of the number of slices containing incoming flux information contained within the ghost cells, and for each of these slices, it will build a slice flux communication structure, and store these in a two dimensional vector organized according to the patch on which their inlet face resides.

After the local sweep has been performed, the thread then enters the second critical region. In this critical region, the slice flux communication structures gathered during the local sweep are added to a three dimensional vector which collects these structures accumulated by all tasks, and organizes them according to the angle index and patch index on which their inlet face resides. Since it is possible that two different tasks could end up with two slice flux communication structures belonging to the same slice on the node sharing the patch, care must be taken not to overwrite one with the other, and instead merge them appropriately.

In the next step, while still inside the second critical region, the number of slice flux communication structures collected during the local sweep is added to a slice counter organized according to the angle index and patch index on which their inlet face resides. It is then checked whether this count matches the total number of slices which would constitute a complete outlet patch, in order to determine whether a communication should be made. This total number of slices is determined at initialization, for example as a by-product of testing the mesh for slicing as should be done before beginning the simulation. In addition to checking whether the outlet patch has a complete set of slice flux communication structures, it also must ensure that the outlet patch has not previously been completed and communicated. If these criteria are met, the outlet patch-angle pair is marked as complete so that no other thread will subsequently try to communicate the patch-angle pair, and then the current thread sends this patch's worth of slice flux communication structures to the node sharing the outlet patch, using the angle index as a label for the communication.

Finally, after exiting the second critical region, and before returning to the beginning of the task loop, a final check is made to see if the current task index is the final task index, and if so, whether all tasks have been executed. If so on both counts, the thread will break from the task loop, and the global sweep is finished once all threads reach either this point, or the similar check at the beginning of the task loop. Otherwise, the task loop index is reset yet again.

It should be noted that the critical regions, which take up much of Algorithm 4.1, comprise a minuscule amount of the actual work being performed. By far, the overwhelming majority of the work takes place in the local sweep in between the two critical regions. Thus, the greatest efficiency is attained when all threads are able to quickly be assigned tasks ready for execution, and proceed to perform local sweeps in unison. The heuristic ordering and the frequent resetting of the task loop index ensures that the highest priority tasks are completed first so that tasks on other nodes become ready for execution as quickly as possible, and this tends toward maximizing the efficiency.

4.2 Parallelization Option 2

The second parallelization option presented here strays further from the traditional transport sweep than the first parallelization option. This second option was developed with two goals in mind; to eliminate the need for planar inter-node domain boundaries, and to eliminate idle time altogether. The first of these goals is important because problem geometries rarely contain such natural planar divisions, and even if they did, it is not guaranteed that these planes would be located such that acceptable load-balancing metrics would be achieved. While planar divisions can be achieved with acceptable load-balancing metrics as seen in the previous section, depending on the mesh type this can introduce large numbers of poor-quality cells and faces on the inter-node domain boundaries and increase the number of cells in the global mesh significantly. Thus the brick decomposition and load-balancing algorithm alluded to in the previous section is by no means a panacea. The second goal is important because it removes a stringent upper limit on the parallel efficiency.

4.2.1 Description

The second parallelization option aims to separate the connection between where the solution is stored, and where it is obtained. For instance, just because node 1 "owns" a particular cell, node 1 does not necessarily have to calculate the solution on that cell. This allows an arbitrary domain decomposition of the mesh onto nodes for which to store the solution, and hence there is no longer a need for planar inter-node domain boundaries. This also allows for a wealth of domain decomposition strategies such as the Scotch algorithm[24], which have excellent load-balancing properties, and are implemented in open-source software packages. Figure 4.4 shows the mesh in Figure 4.1a decomposed using the Scotch algorithm into 8 sub-domains.

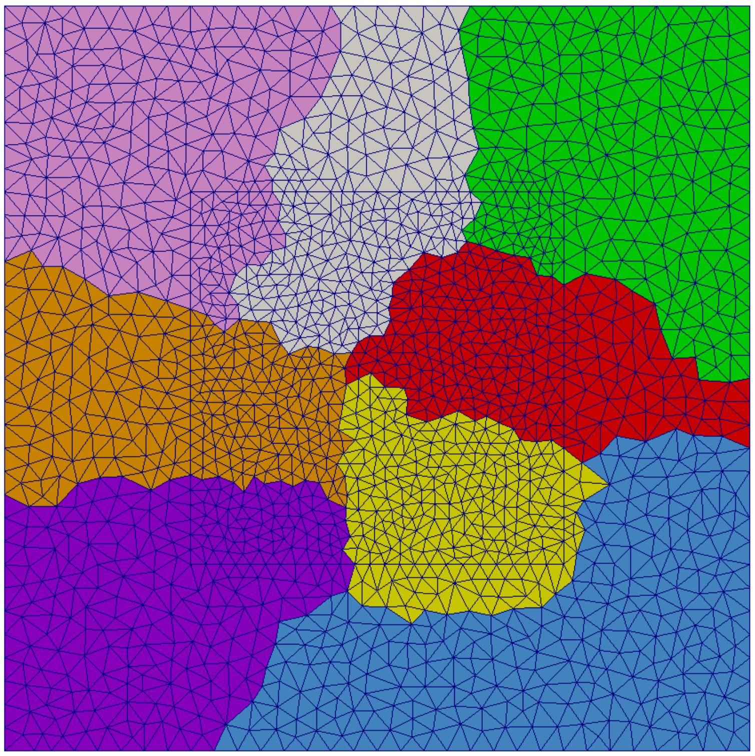


Figure 4.4: Scotch decomposition of the mesh depicted in Figure 4.1a.

Once we have decomposed the mesh such that each node owns roughly the same number of cells for which to store the solution, we can then proceed by drawing cut planes through the entire global mesh for a given angle. These cut planes can be positioned such that there are roughly the same number of slices to be solved within each region, which will be called pipes. This is illustrated in Figure 4.5. The solution in each pipe for the depicted angle would then be performed by a single node, without any communication occurring during the sweep. Furthermore, since the solution in each angle is independent of the solution in any other angle, several angles can be solved for simultaneously, each with its own pipe decomposition. For instance, if there were 16 nodes, 2 angles could be solved simultaneously, each with a decomposition consisting of 8 pipes.

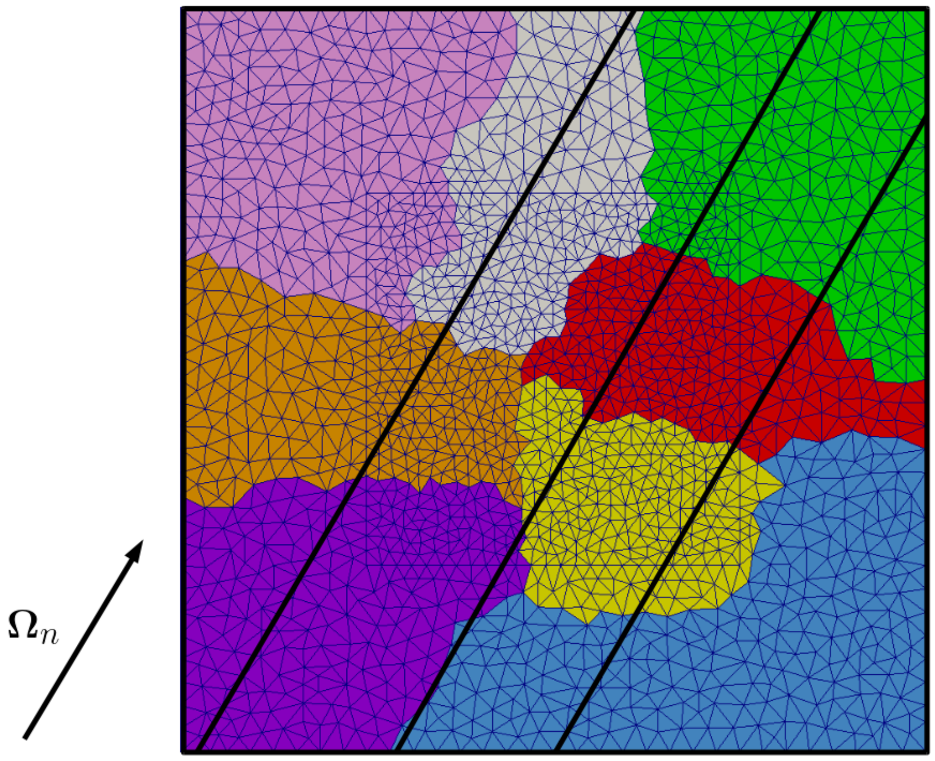


Figure 4.5: Pipe decomposition of Scotch decomposed mesh into 4 pipes.

While no communication is necessary during each sweep, communication before and after the sweep is necessary. With an arbitrary domain decomposition where each node is assigned a region of the mesh for which to store the solution, each node must communicate the source moments as well as the total cross section in each of its local cells to the nodes whose pipes contain these cells. The nodes containing these cells in their pipes will have to communicate the contribution to the solution for these cells back to the node that owns these cells when the sweep is completed.

In this way, volumetric information is communicated between nodes instead of boundary information, resulting in larger messages, but hopefully fewer of them. This minimizes the cost of message passing latency which is typically orders of magnitude higher than the cost of per byte communication. One notable drawback is that the entire mesh geometry must be known by every node in order to construct the slices and sub-slices within its pipe. The mesh can be described by the point coordinates, point indices on each face, face indices on each cell, cell centroids and volumes, and face normal vectors. This amounts to roughly 400 bytes per cell, depending on the mesh type and complexity; however the global mesh is still likely orders of magnitude smaller than the full transport solution on each subset of the mesh.

4.2.2 Algorithm

The global sweep algorithm for parallelization option 2 is shown in Algorithm 4.2, which is to be executed on each node. This operation proceeds as a loop over angle-sets, which are the angles to be swept simultaneously. For instance, with 16 nodes we could use angle-sets of size 1, 2, 4, 8, or 16, with each angle consisting of a pipe decomposition of 16, 8, 4, 2, or 1 pipes respectively. Within each iteration of the angle-set loop, each node is responsible for performing a sweep for a prescribed pipe-angle pair.

Algorithm 4.2: Global transport sweep for parallelization option 2.

```
1: **for** $a = 0$ to $N_{\text{angle-sets}} - 1$ **do**
2:     localCellSources = 2D vector of sourceStructs of dimension
3:                        $N_{\text{local cells}} \times (N_{\text{angles per angle-set}} \times N_{\text{groups per group-set}})$
4:     **for** $i = 0$ to $N_{\text{local cells}} - 1$ **do**
5:         **for** $j = 0$ to $N_{\text{angles per angle-set}} - 1$ **do**
6:             **for** $b = 0$ to $N_{\text{groups per group-set}} - 1$ **do**
7:                 $m = a \times N_{\text{angles per angle-set}} + j$
8:                 $l = j \times N_{\text{groups per group-set}} + b$
9:                 $s$ = source-set index($i$)
10:                $g$ = energy group index($b$)
11:                localCellSources[$i$][$l$].gci = global cell index($i$)
12:                localCellSources[$i$][$l$].$g = g$
13:                localCellSources[$i$][$l$].$\sigma_{t,g}$ = total cross section($i$, $g$)
14:                localCellSources[$i$][$l$].$q^c, q^x, q^y, q^z$ = ComputeSource($i$, $m$, $s$, $g$)
15:            **end for**
16:        **end for**
17:    **end for**
18:    indexInSendTo = vector on ints of length $N_{\text{nodes}}$
19:    **for** $j = 0$ to sendTo[$a$].size $-1$ **do**
20:        indexInSendTo[sendTo[$a$][$j$]] = $j$
21:    **end for**
22:    sourceSendVec = 2D vector of sourceStructs of dimension
23:                    (sendTo[$a$].size +1) $\times N_{\text{sources going to nodes in sendTo}[a]\text{ entries}}$
24:    **for** $i = 0$ to $N_{\text{local cells}} - 1$ **do**
25:        **for** $j = 0$ to $N_{\text{angles per angle-set}} - 1$ **do**
26:            **for** $b = 0$ to $N_{\text{groups per group-set}} - 1$ **do**
27:                $m = a \times N_{\text{angles per angle-set}} + j$
28:                $l = j \times N_{\text{groups per group-set}} + b$
29:                **for** $k = 0$ to $N_{\text{pipes per cell}}[i][m] - 1$ **do**
30:                    $p$ = pipeIndices[$i$][$m$][$k$]
31:                    $n$ = nodeIndices[$m$][$p$]
32:                    **if** $n \neq$ myRank **then**
33:                        $s$ = indexInSendTo[$n$]
34:                    **else**
35:                        $s$ = sendTo[$a$].size
36:                    **end if**
37:                    append localCellSources[$i$][$l$] to sourceSendVec[$s$]
38:                **end for**
39:            **end for**
40:        **end for**
41:    **end for**
```

```
for $i = 0$ to sendTo[$a$].size $-1$ do
    send sourceSendVec[$i$] to sendTo[$a$][$i$]
end for
mySources = vector of sourceStructs
for $i = 0$ to recvFrom[$a$].size $-1$ do
    receive vector of sourceStructs from recvFrom[$a$][$i$] into
                    a vector named tempSources
    for $j = 0$ to $N_{\text{tempSources}}$ do
        append tempSources[$j$] to mySources
    end for
end for
for $i = 0$ to sourceSendVec[sendTo[$a$].size].size $-1$ do
    append sourceSendVec[sendTo[$a$].size][$i$] to mySources
end for
myFluxes = vector of fluxStructs of length mySources.size $\times N_{\text{angles}}$
$s = 0$
for $i = 0$ to mySources.size $-1$ do
    for $n = 0$ to $N_{\text{angles}} - 1$ do
        myFluxes[$s$].gci = mySources[$i$].gci
        myFluxes[$s$].$n$ = $n$
        myFluxes[$s$].$g$ = mySources[$i$].$g$
        myFluxes[$s$].$\text{LHS}_{3.36c}$,$\text{LHS}_{3.36x}$,$\text{LHS}_{3.36y}$,$\text{LHS}_{3.36z}$ $= 0$
        $s$ += 1
    end for
end for
perform local sweep for slices in cells belonging to mySources,
                    while accumulating cell flux moments into myFluxes
indexInRecvFrom = vector on ints of length $N_{\text{nodes}}$
for $j = 0$ to recvFrom[$a$].size $-1$ do
    indexInRecvFrom[recvFrom[$a$][$j$]] = $j$
end for
fluxSendVec = 2D vector of fluxStructss of dimension
                    (recvFrom[$a$].size $+1$) $\times N_{\text{fluxes going to nodes in recvFrom}[a]\text{ entries}}$
for $i = 0$ to myFluxes.size $-1$ do
    $c$ = myFluxes[$i$].gci
    $n$ = cellOwner[$c$]
    if $n \neq$ myRank then
        $r$ = indexInRecvFrom[$n$]
    else
        $r$ = recvFrom[$a$].size
    end if
```

```
        append myFluxes[$i$] to fluxSendVec[$r$]
    end for
    for $i = 0$ to recvFrom[$a$].size $-1$ do
        send fluxSendVec[$i$] to recvFrom[$a$][$i$]
    end for
    resize myFluxes to zero
    for $i = 0$ to sendTo[$a$].size $-1$ do
        receive vector of fluxStructs from sendTo[$a$][$i$] into
                    a vector named tempFluxes
        for $j = 0$ to $N_{\text{tempFluxes}}$ do
            append tempFluxes[$j$] to myFluxes
        end for
    end for
    for $i = 0$ to fluxSendVec[recvFrom[$a$].size].size $-1$ do
        append fluxSendVec[recvFrom[$a$].size][$i$] to myFluxes
    end for
    for $i = 0$ to myFluxes.size $-1$ do
        $c$ = myFluxes[$i$].gci
        $l$ = globalToLocal[$c$]
        $n$ = myFluxes[$i$].$n$
        $g$ = myFluxes[$i$].$g$
        LHS$_{3.36c}$[$l$, $n$, $g$] += myFluxes.LHS$_{3.36c}$
        LHS$_{3.36x}$[$l$, $n$, $g$] += myFluxes.LHS$_{3.36x}$
        LHS$_{3.36y}$[$l$, $n$, $g$] += myFluxes.LHS$_{3.36y}$
        LHS$_{3.36z}$[$l$, $n$, $g$] += myFluxes.LHS$_{3.36z}$
    end for
    barrier
end for
```

At the beginning of the angle-set loop, each node builds a vector of source communication structures representing the particle source within each of its local cells. Such a structure should contain the information necessary to identify the cell, the energy group, the corresponding total cross section, and the source spatial moments. This should be done by each node on each of its local cells on which it is responsible for storing the solution, because the flux moments from the previous scattering iteration are required to build the scattering source within a given cell. Communicating the flux moments on a volumetric basis would be quite prohibitive, whereas the complete description of the source for a given direction is contained in only four values, namely q^c, q^x, q^y, and q^z.

Once the source spatial moments are computed on each local cell of each node, they must be re-organized in order to send them to the nodes that require this information in order to complete their local sweep in this particular iteration of the angle-set loop. This requires some fairly complicated data structures which are built during initialization, prior to the transport calculation as a by-product of the load balancing step which determines the locations of the cut planes such that each pipe for a given angle contains roughly the same number of slices. The first of these data structures is a list of nodes for which each node must send data to for each iteration of the angle-set loop, referred to in Algorithm 4.2 as sendTo. Similarly, each node holds a data structure named recvFrom which contains a list of nodes for which it must receive data from for each iteration of the angle-set loop.

The next data structure encountered is referred to in Algorithm 4.2 as pipeIndices on line 30. This data structure is essentially a three dimensional vector in which the first two dimensions are $N_{\text{local cells}} \times N_{\text{angles}}$. Each entry in this two dimensional vector is itself a variable length vector containing the pipe indices that each cell falls into for a given angle index. For instance, if the given cell falls within a single pipe for a given angle's pipe decomposition, this vector would be of length one, containing the index of that pipe. If the cell is so large that parts of it lie in three different pipes for a given angle's pipe decomposition, this vector would be of length three, containing the indices of these three pipes.

The last data structure referenced without explanation in Algorithm 4.2 is nodeIndices on line 31. This data structure is essentially a two dimensional vector of dimension $N_{\text{angles}} \times N_{\text{pipes}}$. Each entry in this vector is the node index which is responsible for computing the solution in the pipe-angle pair defined by the given indices. Unlike the other data structures previously mentioned, which were unique on each node, each node stores an identical copy of nodeIndices. These data structures are used to package the source communication structures into vectors based on which node, or set of nodes, each source structure should be sent to. Once these vectors are constructed, they can be communicated easily by sending a single vector of source communication structures between nodes.

With the messages sent, each node loops over the nodes in its recvFrom vector for the particular angle-set index, receives a vector of source communication structures, and appends them to a local vector named mySources. It must also append to this list any sources from cells that the node owns for both storage and computation. Once all messages have been received, each node then builds a vector of flux communication structures, which contain the global cell index, angle index, group index, and the contributions to the left hand sides of equations 3.36, which will be computed during the local sweep in its designated pipe-angle pair. After performing this local sweep, these flux communication structures will be sent back to the nodes owning each cell. To do this, the flux communication structures are re-organized in much the same way that the source communication structures were so that the communication step can be done easily by sending a single vector of flux communication structures between nodes. Once the flux communication structures have returned to the nodes owning their cells, the left hand sides of equations 3.36 are updated, and a barrier is placed before the end of the angle-set loop so that all nodes start the next iteration together.

4.3 GPU Acceleration of the LDFE Extended SBA

Up until now, the theory and implementation of the LDFE spatial discretization into the extended SBA, and the parallelization strategies that such an approach makes possible, have not addressed the legitimate concern of memory requirements. One must remember that the slices and sub-slices of the mesh are unique to each angle in the S_N angular quadrature set, with the exception that the slices for angles in opposite directions are geometrically the same. This means that the number of slices and sub-slices will be on the order of $N_{\text{local cells}} \times N_{\text{angles}}$ for each node.

For angular quadrature sets with thousands of angles and meshes in which each node contains tens or hundreds of thousands of cells, this makes storage and re-use of all quantities unique to each slice or sub-slice unrealistic, and the consequence of this is that these quantities must be re-computed each time they are needed, or more precisely once per transport sweep. As discussed in Chapter 2, while one goal of iterative methods for particle scattering is to reduce the number of sweeps needed to arrive at the numerical solution, for most problems of interest it is unavoidable that a non-trivial number of transport sweeps will be required, and this is the motivation for focusing so much attention on the parallel efficiency of the transport sweep itself.

If we consider the traditional and extended SBA using the LDFE spatial discretization, we can immediately note that many quantities are needed for each slice (and sub-slice in the extended SBA), and that these quantities are mostly geometric ones that can be computed independently in an embarrassingly parallel fashion. Even so, if the time spent computing this information before each sweep is much greater than the time required for the sweep itself, we would essentially be spending most of our time computing the same exact values over and over again. If this is indeed the case, it is clearly not ideal, but perhaps we can look to the GPU to remedy this. First, let us state concretely the necessary quantities that must be computed and stored throughout the sweep for each slice and sub-slice. We begin with the quantities required for each slice, which have been tabulated in Table 4.1, assuming 1 byte per boolean, 4 bytes per integer, and 8 bytes per float.

Table 4.1: List of quantities required to compute on each slice of the mesh.

Quantity	Purpose	Bytes
c, f_{in}, f_{out}	identifying information	12
$\overline{x}_s$, $\overline{y}_s$, $\overline{z}_s$	for use in eqs. 3.3 - 3.6	24
Δx_s, Δy_s, Δz_s	for use in eqs. 3.3 - 3.6	24
V_s, $A_{s,\text{in}}$, $A_{s,\text{out}}$	basic geometric information	24
$M_{ij}^s \quad ; \quad i,j = x,y,z$	for use in eqs. 3.26 - 3.29	48
$\mathbf{A}$	reduced coefficient matrix of eqs. 3.26 - 3.29	128
D	boolean for if slice is done	1
G	boolean for if slice is in ghost cell	1
$\sigma_{t,s}$	for use in eqs. 3.26 - 3.29	$8 \times \mathcal{N}$[†]
$\psi_{n,s,\text{in}}^i \quad ; \quad i = c,x,y,z$	for use in eqs. 3.26 - 3.29	$32 \times \mathcal{N}$
$q_{n,s}^i \quad ; \quad i = c,x,y,z$	for use in eqs. 3.26 - 3.29	$32 \times \mathcal{N}$
f, S_x, S_y, S_z	communicating flux to downstream sub-slices	$32 \times \mathcal{N}$

[†]$\mathcal{N}$ is the number of groups per group-set in the sweep. In the Gauss-Seidel iterative method, this would be equal to one, while in the Jacobi iterative method, this would be equal to the number of energy groups.

The first entry in Table 4.1 references the identifying information of the slice, namely the inlet and outlet face indices, f_{in} and f_{out}, but also the index of the parent cell c, for the purpose of computing source moments and contributing to the left hand sides of equations 3.36. It could be argued that if this were the only information to be stored permanently in memory about each slice in the mesh, the present discussion would be unnecessary. Indeed, not much more information than this is required in existing implementations of the traditional SBA in which the spatial discretizations are limited to diamond difference and characteristic-like schemes. It is only when a higher order scheme like the LDFE spatial discretization is used that one is confronted with the unfortunate reality that storing all slice-dependent information is simply too costly. It is quite possible however, that storing just this information permanently in memory may improve the performance of the current method significantly, since identifying each slice in the mesh is no small feat to perform before each sweep.

The next two entries in Table 4.1 reference the centroid coordinates and extents of the slice used in the definition of the slice basis functions, equations 3.3 through 3.6. It should be noted that the basis functions could be defined without these quantities, however their definition as given in Chapter 3 simplifies the math involved in computing volumetric integrals over the slice and reduces their round-off error. In other words, the memory could be saved at the expense of more floating point operations and less accuracy in the calculation of volumetric integrals. The next two items in the list are the most basic geometric information for the slice (the volume and inlet and outlet areas) and the mass matrix entries M^s_{ij} for $i, j = x, y, z$. These quantities are stored in addition to the reduced coefficient matrix $\mathbf{A}$ appearing next on our list, so that the last remaining integral in equation 3.41 can be computed as a simple linear combination of the angular flux variables obtained by solving equations 3.26 through 3.29.

The reduced coefficient matrix $\mathbf{A}$ is simply the coefficient matrix of equations 3.26 through 3.29, minus all terms that are energy group dependent. These would include all terms in which the total cross section appears, which conveniently also contain the M_{ij}^{s}'s. This is done because it will inevitably be the case that one will want to sweep more than one energy group at a time given the considerable amount of effort needed to prepare the slices of the mesh, which are the same for all energy groups. In such a case, we want a base coefficient matrix that we can simply add terms to in order to get the full coefficient matrix for the given energy group and slice index. The next two items in the list are trivial in the discussion of the memory footprint of each slice, and are used in the local sweep to keep track of slices that have been done and slices that are inside ghost cells.

The remaining terms in Table 4.1 are those that are unique for each energy group, and hence must be stored for each slice and each group in the set of groups being swept simultaneously, hereafter referred to as a group-set. These include the total cross section, incoming facial flux moments, and volumetric source moments. In addition, we will need to represent the flux in the slice in linear form

$$\psi_{n,s}\left(\mathbf{r}\right) = f + S_x x + S_y y + S_z z \ , \tag{4.1}$$

in order to evaluate the integrals in equation 3.44 for each sub-slice downstream of each slice. Each of these coefficients is simply a function of the angular flux variables obtained by solving equations 3.26 through 3.29.

We can now focus our attention on those quantities that must be computed for each sub-slice, thus restricting the conversation to the extended SBA. These quantities are tabulated in Table 4.2, again assuming 1 byte per boolean, 4 bytes per integer, and 8 bytes per float. As in Table 4.1, the first entry in Table 4.2 is the identifying information for each sub-slice. These include the inlet and outlet face indices, f_{in} and f_{out}, the parent cell index c, the upstream slice index u, and the parent slice index p. With this identifying information, each slice can build a list of sub-slices that are downstream of the slice, and a list of sub-slices that are contained within the slice.

Table 4.2: List of quantities required to compute on each sub-slice of the mesh.

Quantity	Purpose	Bytes
c, f_{in}, f_{out}, u, p	identifying information	20
$\overline{x}_{ss}$, $\overline{y}_{ss}$, $\overline{z}_{ss}$	for use in sub-slice basis functions	24
Δx_{ss}, Δy_{ss}, Δz_{ss}	for use in sub-slice basis functions	24
V_{ss}, $A_{ss,\text{in}}$, $A_{ss,\text{out}}$	basic geometric information	24
$\gamma_{ij} \quad ; \quad i,j = c,x,y,z$	for use in evaluating equations 3.44	72
D	boolean for if sub-slice is done	1
G	boolean for if sub-slice is in ghost cell	1
$\psi^i_{n,ss,\text{in}} \quad ; \quad i = c,x,y,z$	for use in evaluating equations 3.45 and 3.46	$32 \times \mathcal{N}$

As in Table 4.1, the next two entries in Table 4.2 reference the centroid coordinates and extents of the sub-slice used in the definition of the sub-slice basis functions, defined similarly to equations 3.3 through 3.6, followed by the most basic geometric information for the sub-slice. The next quantities in the list are the first and second order integrals over the inlet face of the sub-slice

$$\gamma_{ij} = \iint_{\partial V_{ss,\text{in}}} x_i x_j \, d^2 r \ , \tag{4.2}$$

where x_i and x_j are replaced with all combinations of $1, x, y$, and z, resulting in nine values, since γ_{cc} is simply the sub-slice inlet area. These are required because the result of using equation 4.1 for $\psi_{n,ss}(\mathbf{r})$ in equation 3.44 is a linear combination of the γ_{ij}'s. The next two items in Table 4.2 are analogous to their counterparts in Table 4.1. Finally, the values resulting from the evaluation of equation 3.44 must be stored in order to evaluate the incoming flux to the parent slice via equations 3.45 and 3.46.

Now that we have an exhaustive list of the quantities required for each slice and sub-slice, we can discuss how to obtain them, and specifically how to organize these tasks in a way to utilize the GPU most effectively. As stated in Chapter 1, a GPU is most effective at single instruction, multiple data (SIMD) algorithms, in which the same function (or kernel in GPU coding terminology) is applied to each item in a large data set. This is because GPU's are essentially vector processors in which groups of cores of the GPU perform the same operation simultaneously on different data elements. For this reason, it is proposed here to organize the work that must be done during each local sweep into the following eight functions

1. `count_slices` (GPU target)
2. `build_slice_bases`
3. `slice_integration` (GPU target)
4. `count_sub-slices` (GPU target)
5. `build_sub-slice_bases`
6. `sub-slice_integration` (GPU target)
7. `assign_downstream_and_contained`
8. `stages`

where the functions that have the most potential to benefit from GPU acceleration are identified. Those that are not thus labeled are either not SIMD or are likely to take so little time on the CPU that GPU acceleration would be unnecessary.

We begin with the function `count_slices` which identifies all of the slices in the mesh for the given ordinate. Since slices can be identified for each cell in the mesh in an embarrassingly parallel fashion, this task seems well-suited for the GPU where each kernel function is given a cell index to work on. To identify the slices in the cell, we first loop over the faces of the cell and determine if each face is an incoming face for the given ordinate. If it is an incoming face, we again loop over all the faces of the cell and determine if each face is an outgoing face. If it is an outgoing face, we then use the Separating Axis Theorem (SAT)[25] in a two dimensional coordinate system to which the given ordinate is perpendicular, in order to check if the faces overlap. If so, we have found a slice, and its identifying information is stored.

The next function in the list, `build_slice_bases`, is relatively simple. It essentially just takes the information returned from the `count_slices` function and uses it to initialize the slice objects with their identifying information. This function is so simple that it its run-time is expected to be negligible compared to those of the other functions in the list, and hence has not been targeted for acceleration by the GPU. While the action of this function could have been performed inside the previous function, separating the two functions makes the GPU implementation of `count_slices` far simpler.

With the slice objects initialized with their identifying information, the next step is to compute the geometric quantities in Table 4.1, which include the centroid coordinates, extents, volume, inlet and outlet areas, mass matrix entries, and reduced coefficient matrix entries. This is performed by the `slice_integration` function. The first step is to find the vertices of the inlet and outlet faces of the slice. This is done by translation to a two-dimensional coordinate system to which the given ordinate is perpendicular, followed by the application of the Sutherland-Hodgman algorithm for polygon clipping[26] in order to find the intersection of the inlet and outlet faces. This locates the vertices in this rotated two-dimensional coordinate system, which can then be projected back onto the planes in which the inlet and outlet faces reside. Once the vertices of the slice are obtained, the surface and volume integrals can be calculated analytically or with quadrature integration capable of integrating the required polynomials exactly. Since this can be done for each slice in an embarrassingly parallel fashion, this task seems well-suited for the GPU where each kernel function is given a slice index to work on.

Next in the list is the `count_sub-slices` function. This is the sister function to the `count_slices` function, and indeed works quite similarly. Since the sub-slices can be identified from the outlet of each slice in an embarrassingly parallel fashion, this task seems well-suited for the GPU where each kernel function is given a slice index to work on. To identify the sub-slices downstream of each slice, we first must have the vertex coordinates of the slice outlet, and the index of the downstream cell. We then loop over all the faces of the cell and determine if each face is an outgoing face. If it is an outgoing face, we again use the SAT in a two dimensional coordinate system to which the given ordinate is perpendicular, in order to check if the face overlaps with the slice outlet. If so, we have found a sub-slice, and its identifying information, excluding its parent slice index, is stored.

Next we encounter the `build_sub-slice_bases` function, which is again quite similar to its sister function `build_slice_bases`. This function simply initializes the sub-slice objects with their inlet and outlet face indices, parent cell index, and upstream slice index. Its run-time is again expected to be negligible, and as a result it is not targeted for acceleration by the GPU. It is also the case that in a pure

CPU implementation, the action of this function would have been performed in the `count_sub-slices` function.

The next step is to calculate the geometric quantities in Table 4.2, which include the centroid coordinates, extents, volume, inlet and outlet areas, and the first and second order integrals over the sub-slice inlet. This is performed by the `sub-slice_integration` function, which is almost identical to its sister function `slice_integration`, with fewer geometric quantities to compute. Since this task can be performed for each sub-slice in an embarrassingly parallel fashion, this task seems well-suited for the GPU where each kernel function is given a sub-slice index to work on.

With the slices and sub-slices formed, the only work left to do before performing the local sweep is to assign parent indices to each sub-slice, build a list of sub-slices contained within each slice, and build a list of sub-slices downstream of each slice. This action is performed by the next function in the list, `assign_downstream_and_contained`. It works by looping over the sub-slices and adding the sub-slice index to the downstream list of its upstream slice. While inside this loop over the sub-slices, it also loops over the slices within the parent cell of the sub-slice, and checks each one for a matching inlet and outlet face index pair. When a match is found, the slice index is stored as the sub-slice's parent slice, and the sub-slice index is added to the list of sub-slices contained within the slice. This function is not targeted for GPU acceleration because its run-time is expected to be negligible compared to those of the other functions in the list.

The final function on the list, `stages`, is where the actual local sweep is performed. For details on this local sweep, please refer to Algorithm 3.1. This function is not targeted for GPU acceleration because it is not SIMD, and the prospects for making it SIMD are not promising. This does not mean the GPU needs to sit idly by. One could imagine a pipe-lining strategy where the GPU is tasked with preparing the slices and sub-slices for the next task in the task list while this local sweep is being performed by the CPU. If the time for the GPU to prepare the slices and sub-slices is less than the time for the CPU to perform the local sweep, this means the time required to prepare the quantities that could not be stored in memory could be effectively hidden altogether. Hence, it could be the case that the LDFE spatial discretization applied to the traditional and extended SBA, may only be feasible on the next generation of super-computers, where each node has at least one GPU and the burden of having to recompute the quantities in Tables 4.1 and 4.2 for each sweep can be effectively hidden.

5. RESULTS AND ANALYSIS

The research described in the preceding chapters has culminated in the development of a computer code named Slice-T. In this chapter, results generated using this code will be presented. The Slice-T code provides the user with an option to perform transport sweeps using the cell balance approach (CBA), slice balance approach (SBA), or extended slice balance approach (ESBA). When using the ESBA, the user also has the option to choose between the two parallelization strategies presented in Chapter 4. Outer iteration options include Gauss-Seidel and Jacobi iteration, while the inner iteration method is limited to source iteration. The input file and mesh are given in OpenFOAM[27] format. While Slice-T was not written using the OpenFOAM libraries, the choice of OpenFOAM format for the input to the Slice-T code was chosen due to the large user base of OpenFOAM, as well as the wealth of pre-processing and post-processing utilities provided with the OpenFOAM software. All transport cross-sections are supplied in the matxs format provided by the NJOY nuclear data processing code[28], developed by Los Alamos National Laboratory.

To begin this chapter, we will focus on the accuracy of the CBA, SBA, and ESBA. For the SBA and ESBA, we will apply both the linear discontinuous finite element (LDFE) and diamond difference (DD) spatial discretization schemes, while for the CBA we only apply the LDFE spatial discretization scheme. After discussing the accuracy of the methods, we will discuss the parallel efficiency of the parallelization options presented in Chapter 4 for the ESBA, while for the CBA and SBA we will apply a volumetric decomposition parallelization strategy. As transport is an inherently memory bound problem due to the large number of dimensions in the solution's phase-space, we will primarily be concerned with the weak scaling of the parallelization strategies. We will then examine the GPU acceleration of the slice and sub-slice generation process in order to make the LDFE spatial discretization scheme in the SBA and ESBA more viable, and in the process take a closer look at the GPU architecture. Finally, in order to demonstrate the efficacy of the Slice-T code, we will apply the ESBA using the LDFE spatial discretization to a real-world problem, as well as a not-so-real-world problem.

5.1 Accuracy

The accuracy of a given method is largely determined by the rate at which the numerical solution approaches the exact solution as the spatial resolution of the mesh is increased. If the exact solution is known, various metrics can be used to measure this error. The metrics used here are as follows

$$\epsilon_{L^2} = \sqrt{\iiint\limits_{D} \left(\phi_c\left(\mathbf{r}\right) - \phi_e\left(\mathbf{r}\right)\right)^2 \, d^3r} \,, \tag{5.1}$$

$$\epsilon_{\langle L^2 \rangle} = \sqrt{\sum_{i=1}^{N_c} V_i \left(\langle \phi_c \rangle_i - \langle \phi_e \rangle_i\right)^2} \,, \tag{5.2}$$

$$\epsilon_R = \frac{\sum\limits_{i=1}^{N_c} V_i \left(\langle \phi_c \rangle_i - \langle \phi_e \rangle_i\right)}{\iiint\limits_{D} \phi_e\left(\mathbf{r}\right) \, d^3r} \,, \tag{5.3}$$

where $\phi_c\left(\mathbf{r}\right)$ is the numerical solution for the scalar flux, $\phi_e\left(\mathbf{r}\right)$ is the exact solution for the scalar flux, N_c is the number of cells in the spatial mesh, V_i is the volume of cell i, $\langle \phi_c \rangle_i$ is the volume averaged scalar flux for the numerical solution in cell i, $\langle \phi_e \rangle_i$ is the volume averaged scalar flux for the exact solution in cell i, and D is the spatial domain.

Each of these error metrics should converge as $\epsilon = Ch^n$, where C is some coefficient, h is a measure of the cell size in the mesh, and n is the convergence order. The most strict measurement of the error among these metrics is the L^2 norm ϵ_{L^2}. When it is stated that the LDFE spatial discretization is formally second order, it is with respect to a true norm such as ϵ_{L^2} that this statement refers. While the other two metrics are not true norms in the mathematical sense, since they are not derived from a formal inner product of a function space, they do possess significance as it is the cell average flux that is used to compute reaction rates

$$R_i = \iiint\limits_{V_i} \sigma_R \, \phi\left(\mathbf{r}\right) \, d^3r = V_i \, \sigma_R \, \langle \phi \rangle_i \ , \tag{5.4}$$

where R_i is the reaction rate in cell i, σ_R is the response function for the reaction of interest and is assumed constant over the volume of cell i, and $\langle \phi \rangle_i$ is the volume averaged scalar flux in cell i. With reaction rates often computed given the simpler relation on the right side of equation 5.4, one can immediately see the value in measuring the error according to equations 5.2 and 5.3.

In this section, we will measure the convergence rates of each combination of balance approach and spatial discretization scheme for both continuous and discontinuous solutions. For discontinuous solutions, we will also make some qualitative statements about the different combinations in two dimensions, by propagating a single ray of particles, leading to two sharp discontinuities. Finally, we will take a closer look at the DD schemes and explain why the DD schemes appear to be only first order accurate for a smooth solution, given that the DD scheme using the CBA on uniform Cartesian meshes is known to be second order.

5.1.1 Convergence Orders Given a Continuous Solution

In order to measure the convergence rates for the different combinations of balance approach and discretization scheme for continuous solutions, we will need an exact solution for which to compare. This can be obtained via the method of manufactured solutions. The manufactured solution used here is

$$\psi\left(\mathbf{r}\right) = \sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{L}\right)\sin\left(\frac{\pi z}{L}\right) \ , \tag{5.5}$$

for a cubic spatial domain of side length L. A slice through the center of this manufactured solution is shown in Figure 5.1. Also note that the manufactured solution is isotropic, and hence the scalar flux is simply 4π times equation 5.5.

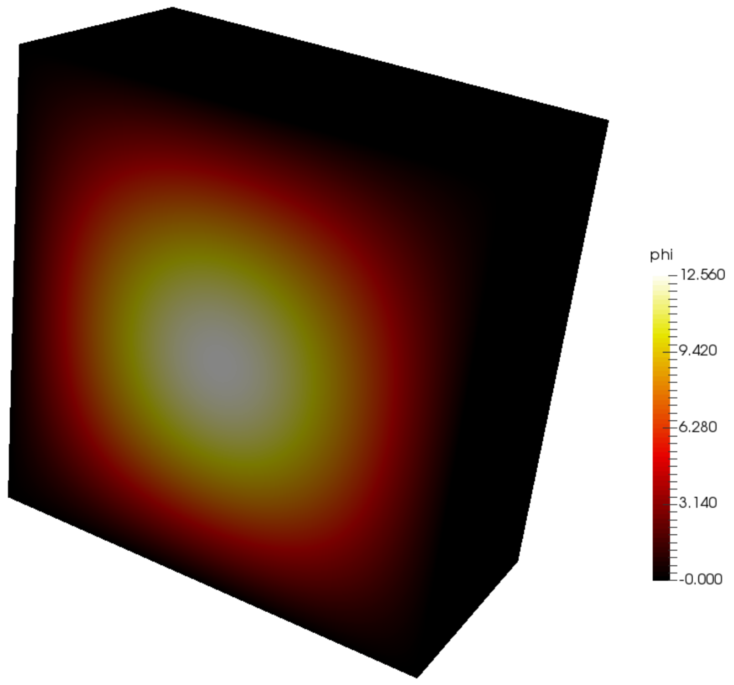


Figure 5.1: Manufactured smooth solution used to measure convergence rates.

We plug this manufactured solution into the steady-state, energy-independent, discrete-ordinates transport equations, assuming isotropic scattering and uniform cross sections

$$\mathbf{\Omega}_n \cdot \nabla\psi_n\left(\mathbf{r}\right) + \sigma_t\,\psi_n\left(\mathbf{r}\right) = \frac{\sigma_s}{4\pi}\phi\left(\mathbf{r}\right) + q_n\left(\mathbf{r}\right)\ , \tag{5.6}$$

in order to derive the source term that leads to this manufactured solution. After a bit of algebra, we end up with

$$\begin{aligned} q_n\left(\mathbf{r}\right) = \frac{\pi}{L}\Big[&\Omega_x \cos\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{L}\right)\sin\left(\frac{\pi z}{L}\right) + \\ &\Omega_y \sin\left(\frac{\pi x}{L}\right)\cos\left(\frac{\pi y}{L}\right)\sin\left(\frac{\pi z}{L}\right) + \\ &\Omega_z \sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{L}\right)\cos\left(\frac{\pi z}{L}\right)\Big] + \\ &\left(\sigma_t - \sigma_s\right)\left(\sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{L}\right)\sin\left(\frac{\pi z}{L}\right)\right)\ . \end{aligned} \tag{5.7}$$

The results that follow use a cubic domain with $L = 200$, uniform material properties, various scattering ratios ($c = \sigma_s/\sigma_t$) up to $c = 0.5$, $\sigma_t = 0.1$, isotropic scattering, an S_4 Gauss-Chebyshev angular quadrature consisting of 32 total angles, and a single energy group. In all of the figures to follow, we will be measuring the above metrics for various mesh resolutions on a uniform Cartesian mesh with $\Delta x = \Delta y = \Delta z = h$. For $h = 1$, this results in a mesh containing 8 million cells.

Figure 5.2 shows the results of using equation 5.1 to measure the error as a function of h. The size of the data points are enlarged for some methods so that the data for one method does not block the data for another. Without this enlargement, only the data last plotted for each discretization scheme would be visible. The only information that can be gleaned from this figure is that all balance approaches using the same spatial discretization scheme have identical convergence rates n, and coefficients C, when using equation 5.1 to measure the error. For the LDFE scheme, all balance approaches are second order as expected. For the DD scheme however, both the SBA and ESBA appear to be only first order, whereas the DD scheme applied to a uniform Cartesian mesh using the standard CBA is known to be second order. This discrepancy will be the topic of a later section.

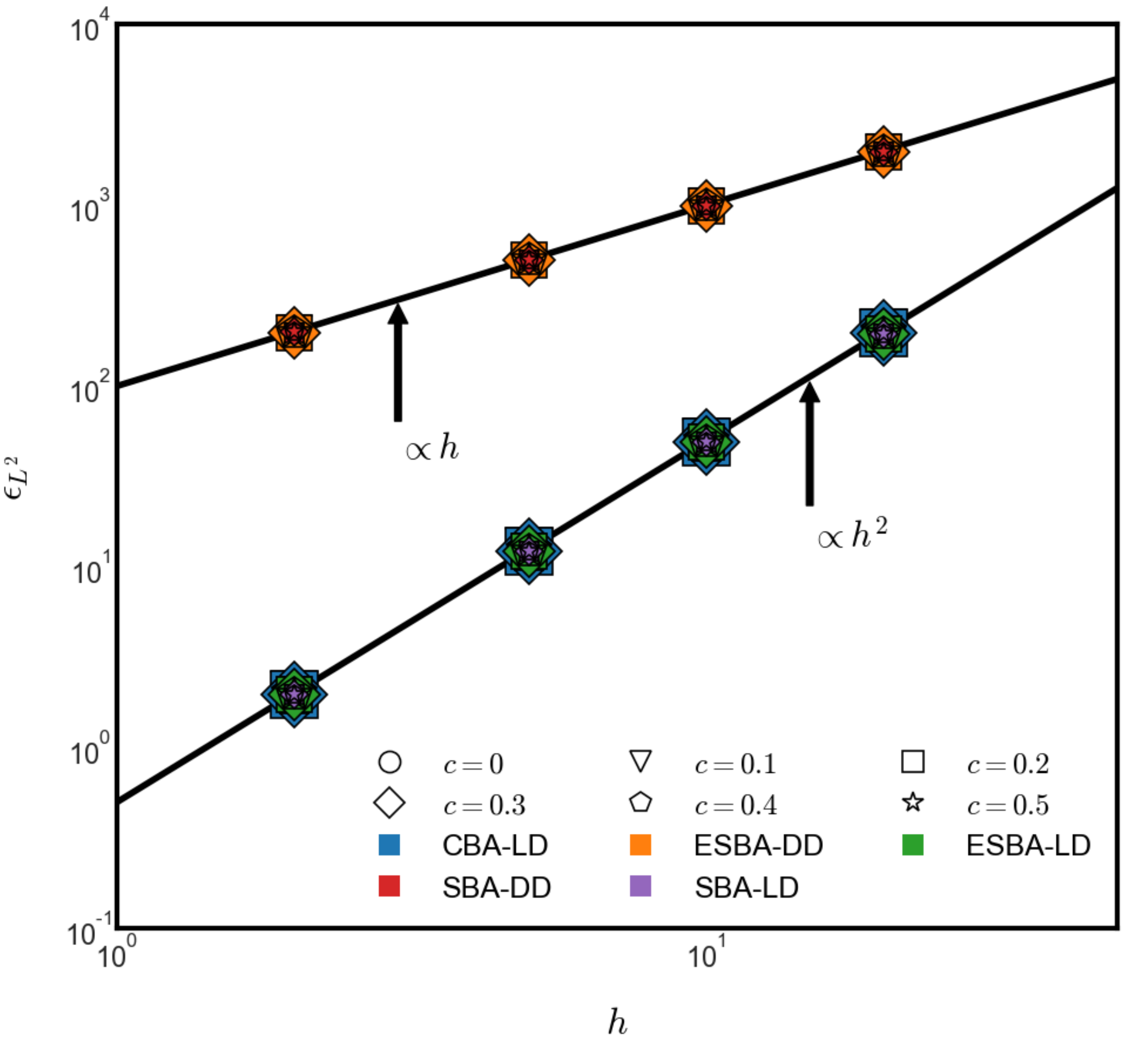


Figure 5.2: Error as a function of cell edge length for the continuous manufactured solution using the error metric given by equation 5.1.

Figure 5.3 shows the results of using equation 5.2 to measure the error as a function of h. Using this error metric, it appears that the DD scheme is first order for both the SBA and ESBA again, however the LD scheme for all balance approaches has a convergence order of 2.6. In addition, it appears that the coefficients are no longer indistinguishable, with $C_{\mathrm{ESBA-LD}} < C_{\mathrm{SBA-LD}} < C_{\mathrm{CBA-LD}}$ and $C_{\mathrm{ESBA-DD}} < C_{\mathrm{SBA-DD}}$. This is the expected behavior, since the ESBA should be more accurate

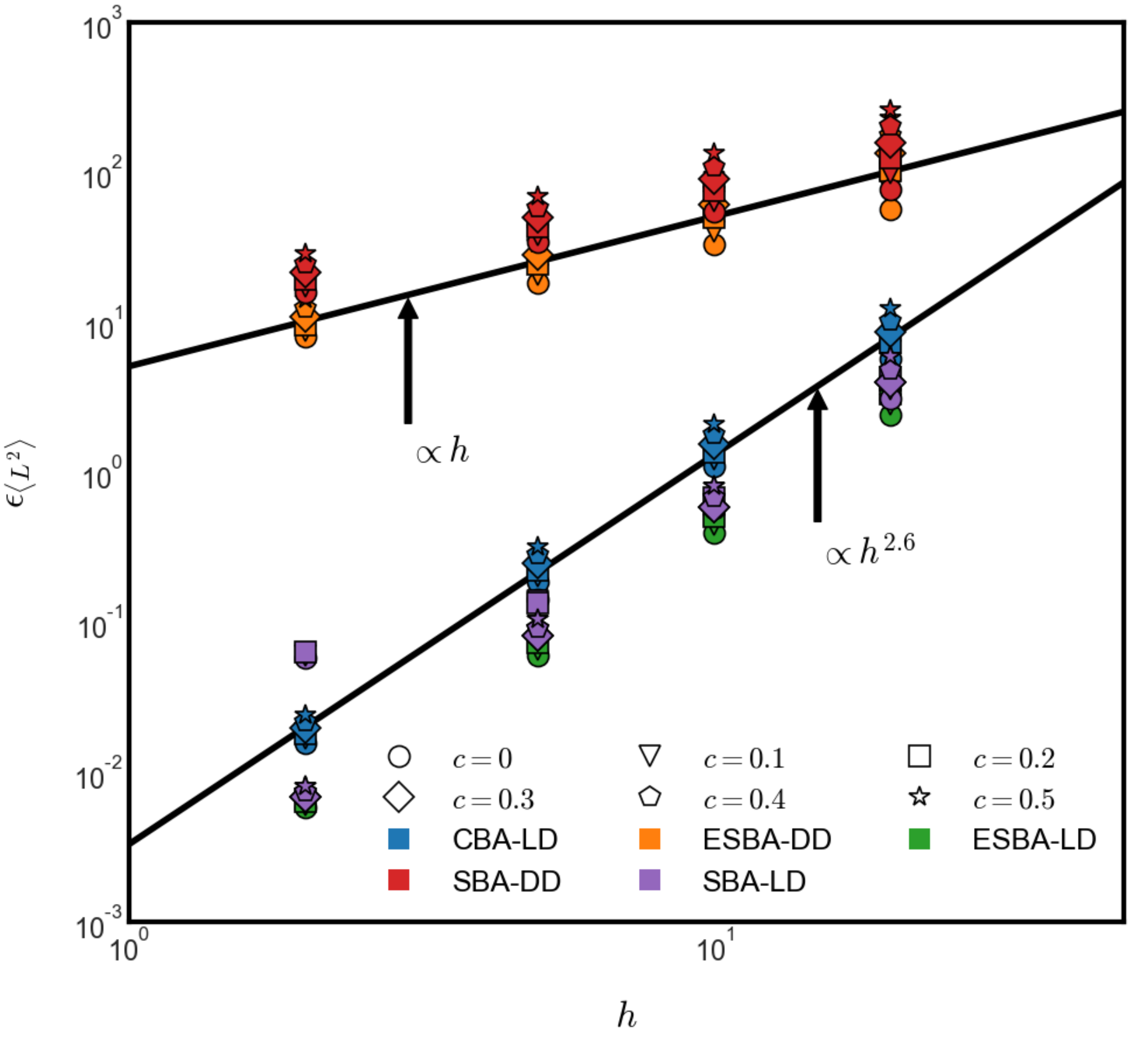


Figure 5.3: Error as a function of cell edge length for the continuous manufactured solution using the error metric given by equation 5.2.

than the SBA, which should be more accurate than the CBA; however, since the solution is smooth, all combinations using the same spatial discretization scheme exhibit the same convergence order.

There are a few outlier data points for the SBA-LD case that are puzzling. These outliers occur only for the lower scattering ratios, which suggests that they are not a result of a coding error or of inadequately converging the scattering iterations. Perhaps we can gain a better understanding of these outliers by examining the final error metric, which is shown in Figure 5.4. In this figure, it appears that all the data points for $c = 0, 0.1$, and 0.2 using the SBA-LD are exhibiting strange behavior. Further, in order to plot the data on a logarithmic scale, we cannot see the sign of the ϵ_R values from Figure 5.4. For completeness, the signs of the ϵ_R values for the SBA-LD case are tabulated in Table 5.1. It should be noted that the sign of ϵ_R is negative for all data points of all other combinations of balance approach and spatial discretization scheme.

It appears that the outlying data points correlate with the cases where the numerical solution over-estimates the exact solution. The cases with higher scattering ratios seem to avoid this issue, perhaps because the solution is being approached from below in smaller steps due to the higher number of scattering iterations needed to converge. We have been unable to find a convincing explanation for the anomalous behavior of SBA-LD for this problem. We note that this research is not focused on the traditional SBA, but on the ESBA, and the ESBA cases are behaving exactly as expected. We therefore leave the solution of the SBA-LD mystery to future work.

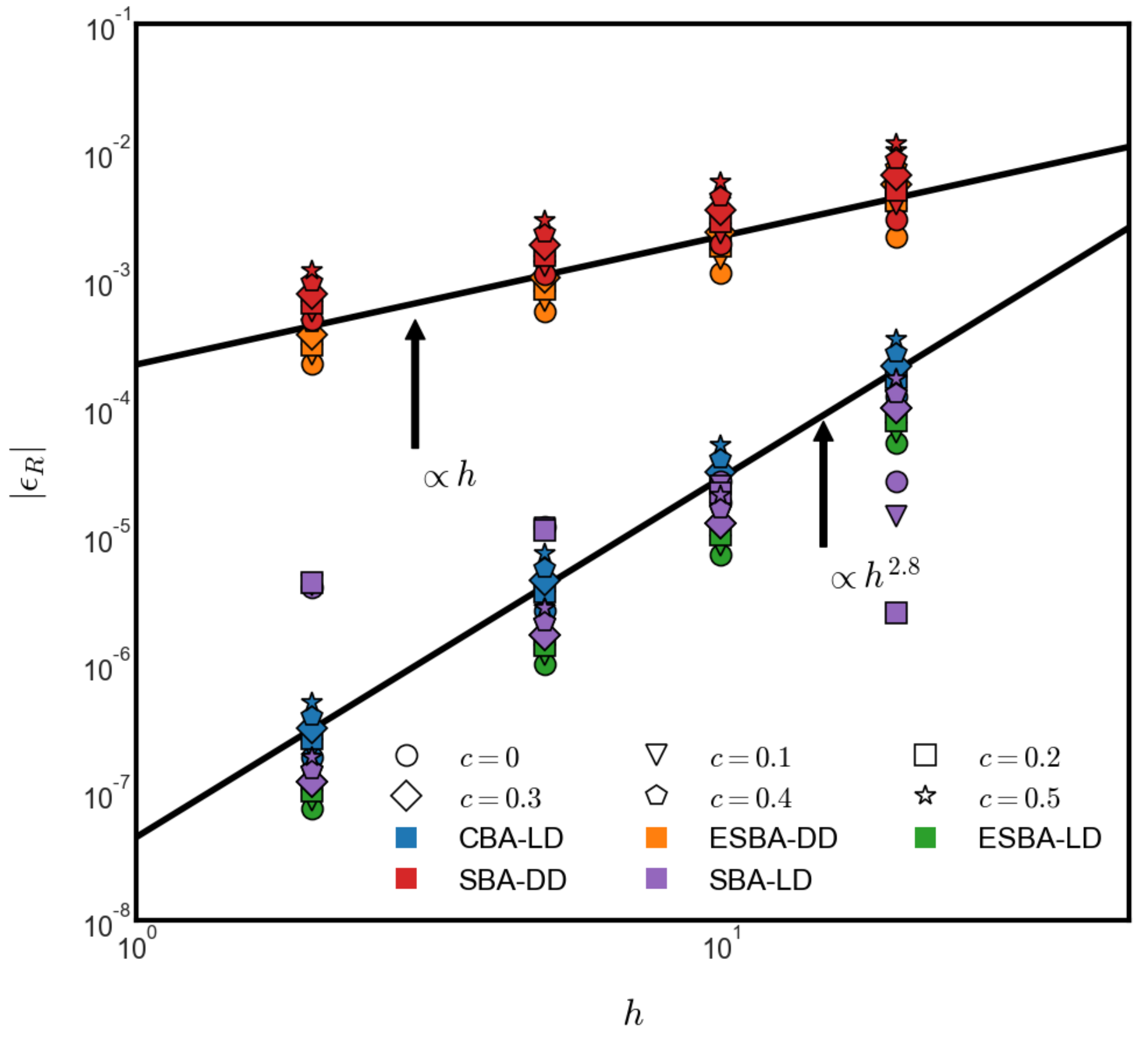


Figure 5.4: Error as a function of cell edge length for the continuous manufactured solution using the error metric given by equation 5.3.

Table 5.1: Sign of the SBA-LD ϵ_R values for the smooth solution convergence study.

h\c	0	0.1	0.2	0.3	0.4	0.5
2	+	+	+	-	-	-
5	+	+	+	-	-	-
10	+	+	+	-	-	-
20	+	+	-	-	-	-

5.1.2 Qualitative Differences in Two Dimensions for a Discontinuous Solution

Before moving on to perform a similar analysis for discontinuous solutions as was done in the previous section, it is instructive to make some qualitative observations of the behavior of the numerical approximations to such solutions. To compare the accuracy of the CBA, SBA, and ESBA for the DD and LDFE spatial discretization schemes, consider the propagation of a single ray in two dimensions. This problem was motivated by Matthews[29] where several finite element, finite volume, nodal, and characteristic methods were compared for their ability to reduce numerical diffusion and lateral oscillations, while also maintaining the correct propagation direction, even for low resolution spatial meshes. While this analysis will not go into nearly as much depth as the cited study, we use the same propagation angle, mesh, and material properties

$$\Omega_x = 0.3500212, \qquad \Omega_y = 0.868890 \ ,$$

$$\Delta x = 0.5, \qquad \Delta y = 0.5 \ ,$$

$$\sigma = 0.02, \qquad \sigma_s = 0 \ ,$$

and compare the solutions for each of the six combinations of balance approach and spatial discretization scheme. The problem examines the propagation of a single ray initiated from a single face on the $x = 0$ boundary nearest the origin. The analytic solution is a decaying exponential ray of width Δx, and zero everywhere else, exhibiting two sharp shadow-type discontinuities. Figure 5.5 shows numerical solutions for the six combinations of balance approach and discretization scheme.

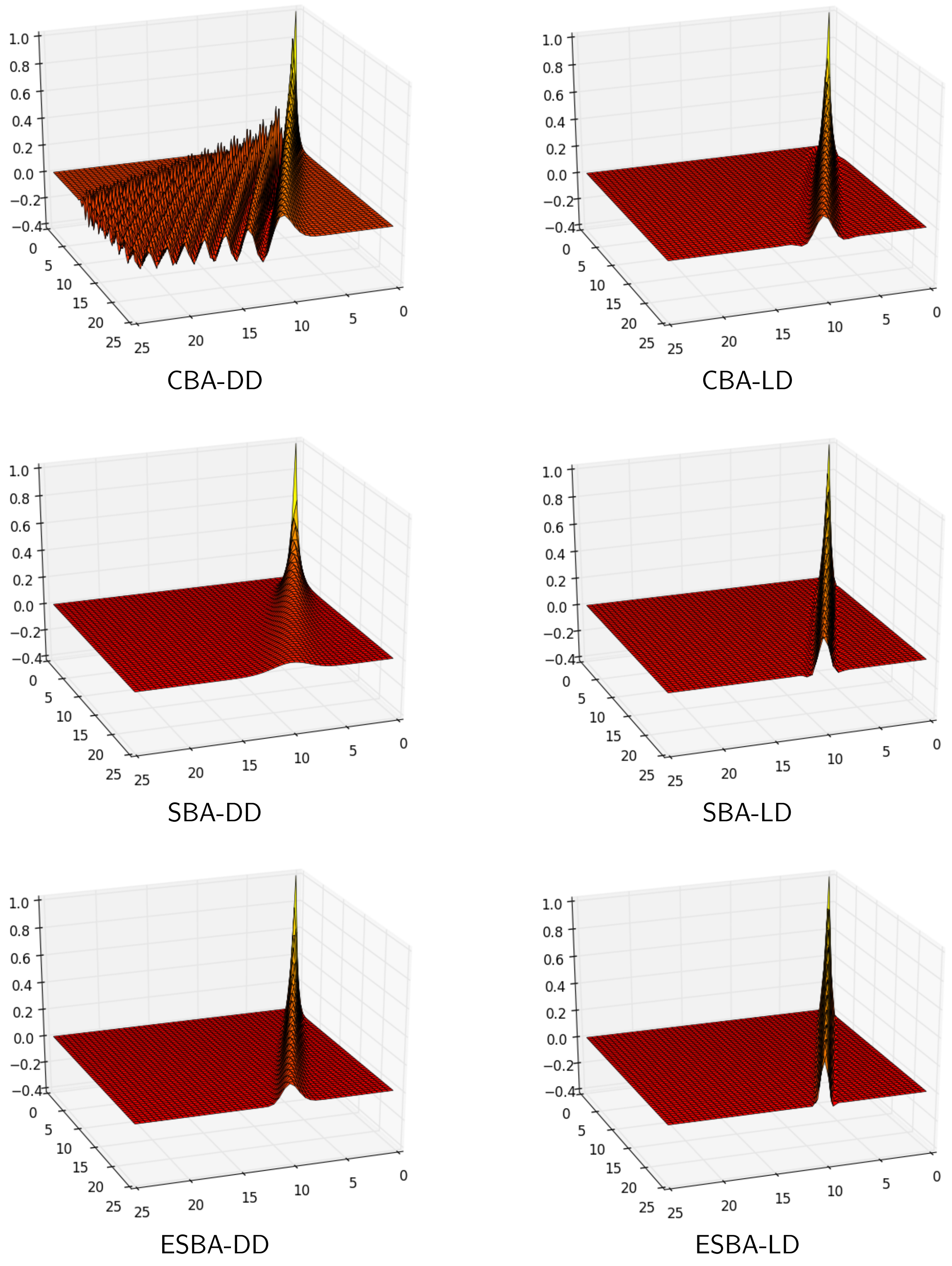


Figure 5.5: CBA, SBA, and ESBA solutions using DD and LDFE spatial discretizations for propagating a single ray in two dimensions.

Figure 5.5 reveals several important results. It shows that the SBA reduces the lateral oscillations present in the CBA for both discretization schemes. This is incredibly effective for the DD scheme due to that methods known propensity for producing large lateral oscillations. These oscillations, and the fact that the CBA-DD combination produces a ray that veers slightly off the propagation direction, led Matthews to recommend that while DD is inexpensive, it is extremely unphysical, and we should abandon it and all of its offspring.[29] This analysis shows that the unphysical oscillations can indeed be fixed with very little extra computation via the SBA.

If we further apply the ESBA to the DD scheme, the results improve even more, producing a sharper ray with less numerical diffusion. Both the SBA and ESBA also steer the DD ray closer to the true propagation direction. The same improvements are obtained for the LDFE scheme; however in this case, the lateral oscillations in the cell balance approach are much less severe, so there is less of a problem to correct. Applying the ESBA to the LDFE scheme produces the sharpest ray of all six, traveling in a direction indistinguishable from the true propagation direction with the given mesh resolution. If the extra computational cost can be afforded, it is clear that the ESBA with the LDFE scheme is the best choice for problems with shadow-type discontinuities.

5.1.3 Convergence Orders Given a Discontinuous Solution

In order to measure the convergence rates for the different combinations of balance approach and discretization scheme for discontinuous solutions, we will need an exact solution for which to compare. In order to accomplish this, we will simply assign an incoming boundary condition for a single angle, to a single boundary patch of the same spatial domain as was used in the continuous convergence study. This produces a decaying exponential in the region of the mesh lying in the projection of the boundary patch in the direction of the chosen angle, and a zero solution elsewhere if we assume no scattering. For the angle, we choose an angle from the S_4 Gauss-Chebyshev angular quadrature set used in the continuous convergence study $\boldsymbol{\Omega}_b = (0.469676, 0.194546, -0.861136)^T$, and for the uniform material properties we choose $\sigma_t = 0.005$. This discontinuous solution can be seen in Figure 5.6. The top boundary patch in the figure that is either solid black or solid red is the boundary patch opposite of where the incoming boundary condition was specified. The incoming boundary condition assigned is $\psi_{\text{inc}}(\boldsymbol{\Omega}_b) = 1.0$.

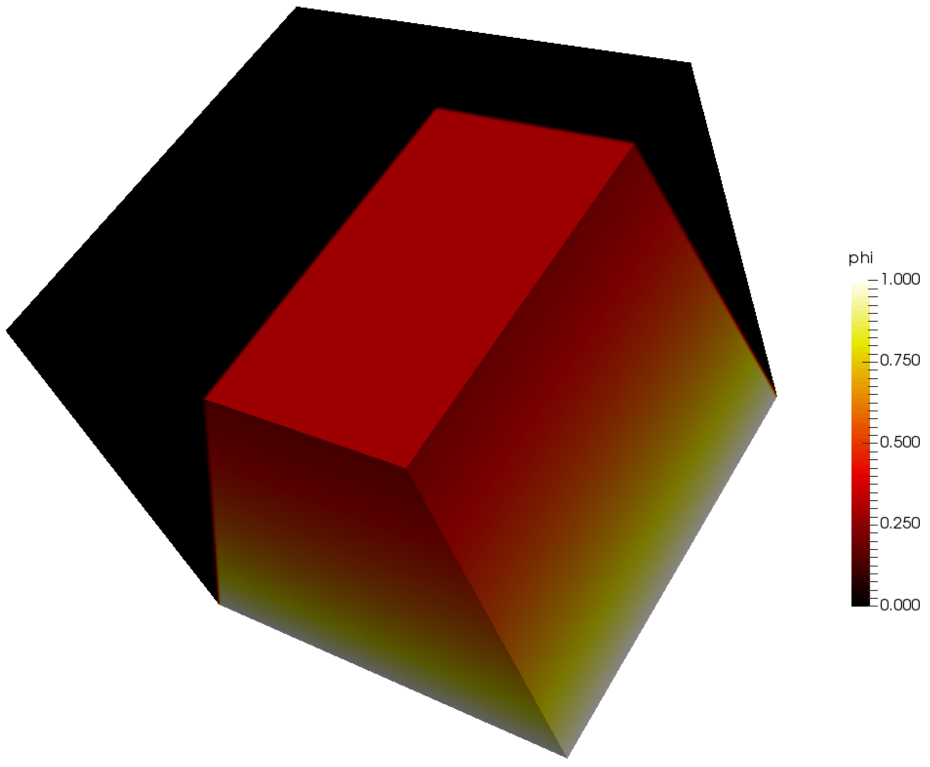


Figure 5.6: Discontinuous solution used to measure convergence rates.

Figure 5.7 shows the results of using equation 5.1 to measure the error as a function of h. The first thing to notice from this figure is that the convergence rates with respect to a formal norm like ϵ_{L^2} have decreased significantly. If we look first at the combinations using the LDFE discretization scheme, we note that $C_{\mathrm{ESBA-LD}} < C_{\mathrm{SBA-LD}} < C_{\mathrm{CBA-LD}}$, as expected, and that the ESBA-LD has a convergence rate of 0.5, while the CBA-LD has a convergence rate of 0.38. The convergence rate of the SBA-LD is not well defined since there appears to be some curvature to the data points, but the error appears to be firmly in between that of the CBA-LD and the ESBA-LD data points. Again, this is to be expected since the ESBA should be more accurate than the SBA, which should be more accurate than the CBA.

If we focus on the two combinations using the DD discretization scheme, we notice that both the ESBA-DD and ESBA-LD exhibit some curvature, so that their convergence rates cannot be clearly identified. While it is still the case that the ESBA appears more accurate than the SBA, as we should expect, the curvature in both is certainly unexpected. It is possible that more data points with $h < 1$ might be required to see the convergence rates clearly. If a line is drawn through the two data points with the smallest values of h for each of the DD combinations, the ESBA-DD appears to have the same convergence rate as the ESBA-LD, while the SBA-DD appears to have the same convergence rate as the CBA-LD.

Discontinuous solutions are known to reduce the convergence rates of numerical methods. Fractional convergence rates are indeed to be expected, but the primary takeaway is that for a discontinuous solution, not only does the ESBA out-perform all other balance approaches in the convergence coefficient as was the case for continuous solutions, but also in the fractional convergence rate. This makes the ESBA-LD the obvious choice when such discontinuities are expected to occur and the computational cost associated with the ESBA can be afforded.

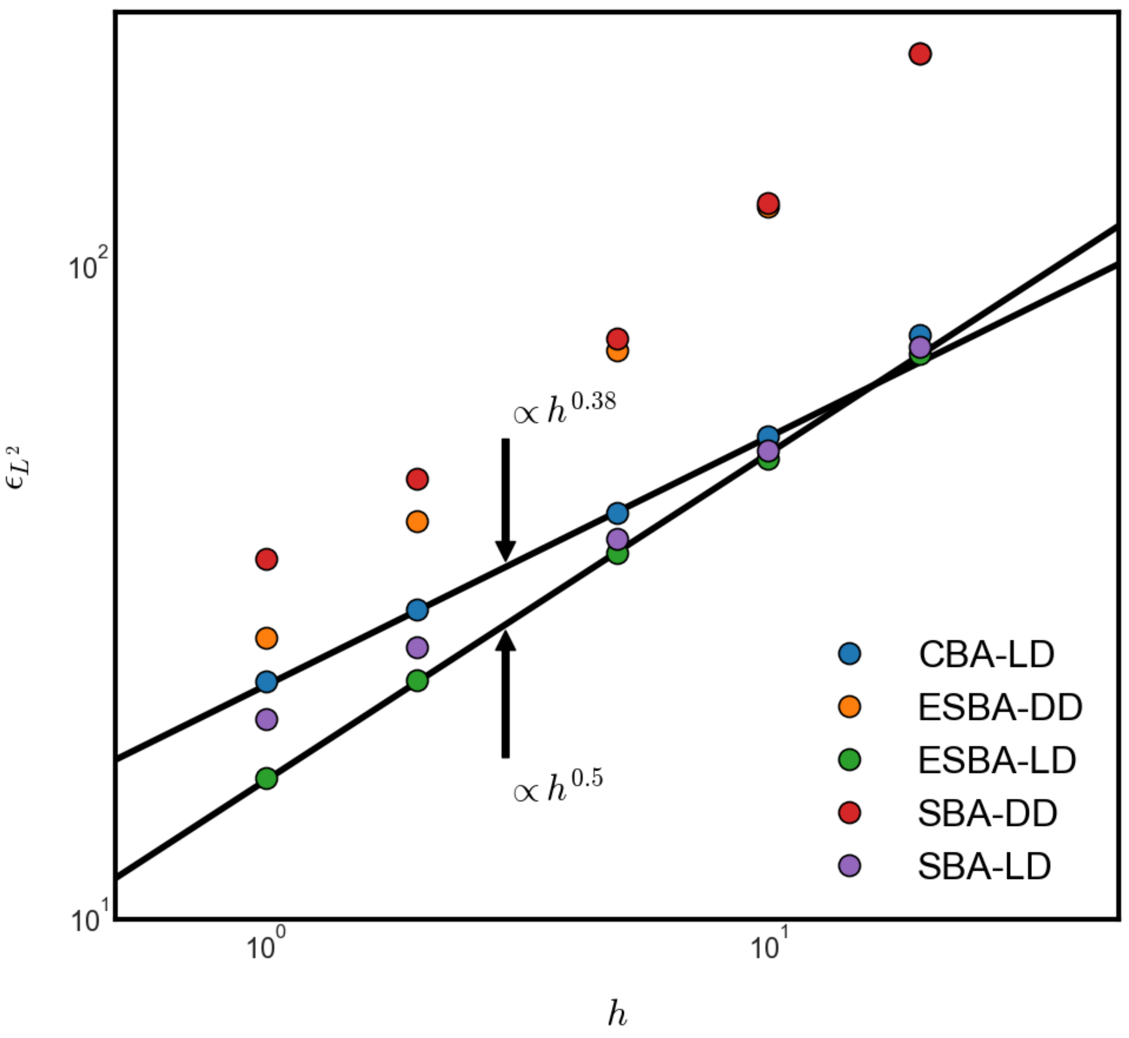


Figure 5.7: Error as a function of cell edge length for the discontinuous solution using the error metric given by equation 5.1.

Figure 5.8 shows the results of using equation 5.2 to measure the error as a function of h. The most interesting thing to note from this figure is that all the CBA and SBA combinations appear not to converge at all, while the ESBA combinations appear to converge at nearly the same rate as they did for continuous solutions. This is a surprising result that we will attempt to explain after we describe the other error measures.

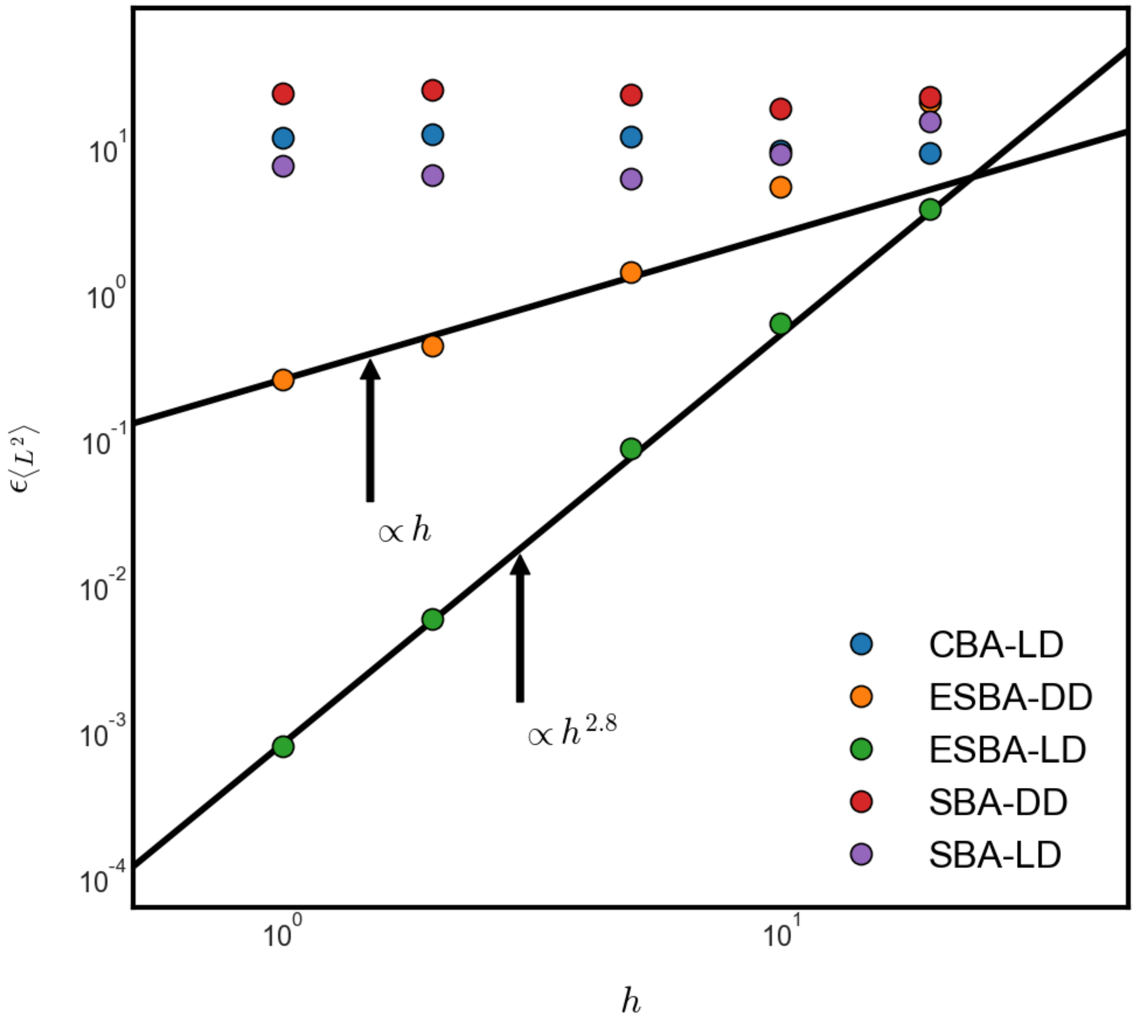


Figure 5.8: Error as a function of cell edge length for the discontinuous solution using the error metric given by equation 5.2.

Figure 5.9 shows the results of using equation 5.3 to measure the error as a function of h. In this figure, all combinations appear to be converging, and even more interestingly, they appear to be converging with the same rate as they did for continuous solutions, with the exception of the SBA-LD. Again, in order to plot this on a logarithmic scale, we are not able to see the signs of ϵ_R. When the sign is analyzed however, the SBA-LD appears to be an outlier from the other combinations in that all of the ϵ_R values are negative, whereas for the four other combinations, all of the ϵ_R values are positive. While we will not venture an explanation for this discrepancy in sign, we shall attempt an explanation for why the non-ESBA combinations appear to converge for ϵ_R, and not for $\epsilon_{\langle L^2 \rangle}$. In addition, it appears that $C_{\text{CBA-LD}} < C_{\text{ESBA-LD}}$, and this is certainly unexpected.

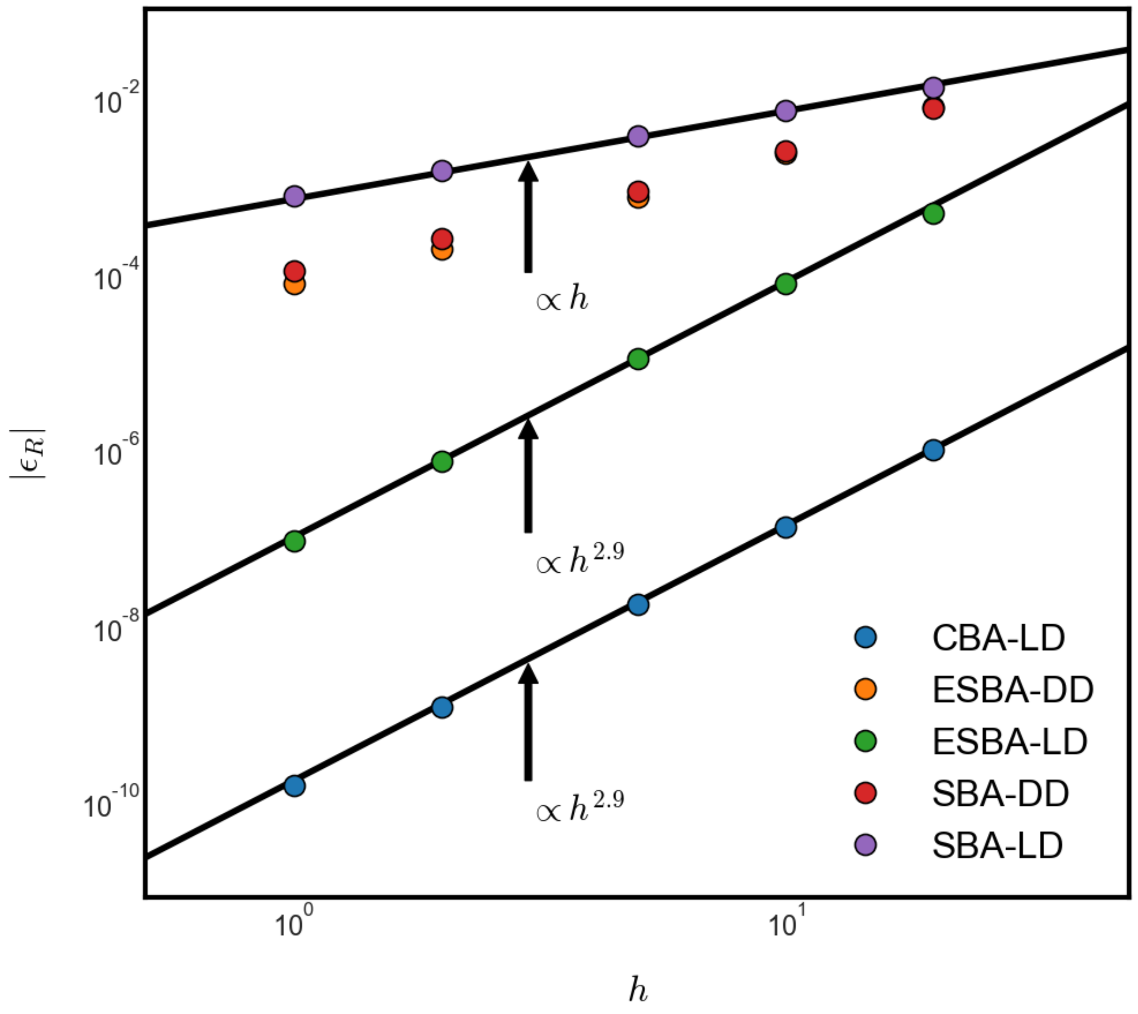


Figure 5.9: Error as a function of cell edge length for the discontinuous solution using the error metric given by equation 5.3.

In order to understand the difference between Figures 5.8 and 5.9, and in particular why $C_{\mathrm{CBA-LD}} < C_{\mathrm{ESBA-LD}}$ in Figure 5.9, we must look again to Figure 5.5. In Figure 5.5, it is apparent that all balance approaches using the LDFE discretization scheme exhibit lateral oscillations to some degree. Since these lateral oscillations cause the numerical solution to oscillate around the exact solution, the ϵ_R error metric, which takes the sign of $(\langle\phi_c\rangle_i - \langle\phi_e\rangle_i)$ into account, is bound to lead to cancellations of the errors in cells where the numerical solution is above the exact solution, with the errors in cells where the numerical solution is below the exact solution. This could lead methods with less damped lateral oscillations to achieve lower values of ϵ_R than methods in which the lateral oscillations are more highly damped, and in particular this would explain why $C_{\mathrm{CBA-LD}} < C_{\mathrm{ESBA-LD}}$ in Figure 5.9.

Of course, a negative value of $\langle\phi_c\rangle_i$ is technically unphysical. It is not possible to have a negative scalar flux. When calculating reaction rates, such values are problematic. An error metric that is strictly positive like ϵ_{L^2} or $\epsilon_{\langle L^2\rangle}$ is more meaningful when negative fluxes are present. The fact that the $\epsilon_{\langle L^2\rangle}$ error metric measures the absolute value of the difference between the numerical and exact solutions may also explain why the CBA and SBA combinations do not appear to converge in Figure 5.8. It is possible that these methods are converging (or at least would be if smaller values of h were included in the study), albeit very slowly, and perhaps we can prove this. We can start by noting that a function $f(\mathbf{r})$ can be written as

$$f(\mathbf{r}) = \langle f\rangle + a(\mathbf{r}) \quad \text{where} \quad \langle a\rangle = 0\ . \tag{5.8}$$

The difference of two functions can then be written as

$$f(\mathbf{r}) - g(\mathbf{r}) = (\langle f\rangle - \langle g\rangle) + (a(\mathbf{r}) - b(\mathbf{r}))\ . \tag{5.9}$$

We can now consider the integral of the square of this difference

$$\iiint_D (f(\mathbf{r}) - g(\mathbf{r}))^2 \ d^3r = \iiint_D ((\langle f\rangle - \langle g\rangle) + (a(\mathbf{r}) - b(\mathbf{r})))^2 \ d^3r = \iiint_D (\langle f\rangle - \langle g\rangle)^2 \ d^3r + 2(\langle f\rangle - \langle g\rangle)\iiint_D (a(\mathbf{r}) - b(\mathbf{r})) \ d^3r + \iiint_D (a(\mathbf{r}) - b(\mathbf{r}))^2 \ d^3r \ . \tag{5.10}$$

Using the fact that the averages $\langle a\rangle = \langle b\rangle = 0$, this can be written as

$$\iiint_D (f(\mathbf{r}) - g(\mathbf{r}))^2 \ d^3r = \iiint_D (\langle f\rangle - \langle g\rangle)^2 \ d^3r + \iiint_D (a(\mathbf{r}) - b(\mathbf{r}))^2 \ d^3r \ . \tag{5.11}$$

Finally, we can treat the integrals over the domain on the right hand side as a sum of integrals over all of the cells in the mesh

$$\iiint_D (f(\mathbf{r}) - g(\mathbf{r}))^2 \ d^3r = \sum_{i=1}^{N_c} V_i (\langle f\rangle_i - \langle g\rangle_i)^2 + \gamma_i \ , \tag{5.12}$$

where

$$\gamma_i = \iiint_{V_i} (a(\mathbf{r}) - b(\mathbf{r}))^2 \ d^3r \ \geq \ 0 \ . \tag{5.13}$$

We can thus state definitively that

$$\iiint_D (f(\mathbf{r}) - g(\mathbf{r}))^2 \ d^3r \ \geq \sum_{i=1}^{N_c} V_i (\langle f\rangle_i - \langle g\rangle_i)^2 \ , \tag{5.14}$$

and hence

$$\epsilon_{L^2} \ \geq \ \epsilon_{\langle L^2\rangle} \ . \tag{5.15}$$

This is indeed the case in Figures 5.7 and 5.8, at least for the ranges of h considered in this study. The result derived above however indicate that if ϵ_{L^2} is converging at a rate of p, then $\epsilon_{\langle L^2 \rangle}$ must be converge at the same rate eventually, which leads one to believe that the $\epsilon_{\langle L^2 \rangle}$ values for the CBA and SBA cases would begin to converge for h values lower than those considered in this study. This convergence however would be at the fractional rate shown in Figure 5.7, and not at the much higher rate obtained by both ESBA combinations in Figure 5.8.

5.1.4 Investigating the First Order Convergence for the SBA and ESBA Diamond Difference Schemes

In order to understand why the DD spatial discretization scheme is only first order in the SBA and ESBA, we should first explain why the DD scheme is second order for uniform Cartesian meshes using the CBA. Consider for instance Figure 2.6, which shows a two dimensional quadrilateral cell. When using the CBA with the DD spatial discretization scheme, there are three variables to solve for on each quadrilateral cell. In Figure 2.6, these are the average flux on the top edge ψ_T, the average flux on the right edge ψ_R, and the cell averaged flux $\langle\psi\rangle$. In addition, we must also keep in mind that for a method to be second order, it must be able produce the exact solution with no error, if the exact solution is linear. The DD scheme for this quadrilateral cell reduces to the following three equations which must must be solved for each cell to calculate the three unknowns

$$\eta_L\psi_L + \eta_R\psi_R + \eta_B\psi_B + \eta_T\psi_T + \sigma_t\langle\psi\rangle = \langle q\rangle \ , \tag{5.16}$$

$$\langle\psi\rangle = 0.5\left(\psi_B + \psi_T\right) \ , \tag{5.17}$$

$$\langle\psi\rangle = 0.5\left(\psi_L + \psi_R\right) \ , \tag{5.18}$$

where ψ_L is the average flux on the left incoming edge and is known, ψ_B is the average flux on the bottom incoming edge and is known, the η's contain the dot product of the edge outward pointing normal with $\mathbf{\Omega}$, times the edge length divided by the cell area, and $\langle q \rangle$ is the cell averaged source. The first equation is simply the transport equation integrated over the cell area, and then divided by the cell area. With some geometric reasoning, it is easy to show that the two closing equations are indeed correct for a linear solution if the quadrilateral cell is rectangular.

We first state without proof that the average of a linear solution along an edge is the value of the linear solution at its midpoint. Similarly, the average of a linear solution over an area is the value of the linear solution at its centroid. For a rectangle, it is also true that the centroid of the rectangle is the midpoint of the line connecting the midpoints of two opposite edges, and the value of a linear function at the midpoint of a line is the average of its value at the line's endpoints, hence equations 5.17 and 5.18.

To be perfectly clear then, the DD scheme in two dimensions using the CBA on a per cell basis is a system of three equations for three unknowns, whose solution is exact if that solution is linear. We should therefore have every reason to expect second order convergence. The method also easily translates to three dimensions in which edges become faces, quadrilateral cells become hexahedral cells, and one more closing equation is added to the system that states that the volume averaged flux is the average of the face averaged fluxes on the front and back faces. We then have four equations with four unknowns, whose solution is still exact if that solution is linear and the hexahedral cells are brick shaped, and is hence still second order. This is then the reason to be concerned with the SBA and ESBA DD schemes exhibiting only first order convergence, but perhaps this can at least be explained.

When the SBA was first proposed by Grove[9], it was postulated that given the form of the balance equation on a slice, simple spatial discretizations that work in one dimension could easily be extended to three dimensions in the SBA framework. Consider for example the following two balance equations, first for a one dimensional cell, and second for a three dimensional slice

$$\frac{\mu_n}{\Delta x_i}\left(\psi_{n,i+1/2} - \psi_{n,i-1/2}\right) + \sigma_{t,i}\psi_{n,i} = q_{n,i}\ , \tag{5.19}$$

$$\alpha_{n,\text{in}}\psi_{n,s,\text{in}} + \alpha_{n,\text{out}}\psi_{n,s,\text{out}} + \sigma_{t,c}V_s\,\psi_{n,s} = V_s\,q_{n,s}\ . \tag{5.20}$$

Recall that these equations appeared in Chapter 2 as equations 2.47 and 2.62. These two equations have very similar forms since $\alpha_{n,\text{in}}$ is negative by definition. Where the DD spatial discretization scheme in one dimension is defined by the addition of the closing equation

$$\psi_{n,i} = 0.5\left(\psi_{n,i-1/2} + \psi_{n,i+1/2}\right)\ , \tag{5.21}$$

the scheme is defined in the SBA framework by the addition of the closing equation

$$\psi_{n,s} = 0.5\left(\psi_{n,s,\text{in}} + \psi_{n,s,\text{out}}\right)\ . \tag{5.22}$$

For the DD scheme in the SBA then, we must solve a system of two equations with two unknowns for each slice. The second equation is not correct for a linear solution, and thus there is no reason to expect second order convergence. This is because for a slice in two or three dimensions, it is not necessarily true that the slice centroid lies at the midpoint of the line connecting the inlet and outlet face centroids. To see this, consider a triangular slice in two dimensions. The centroid of a triangle is the intersection of the triangle's three triangle medians, as shown in Figure 5.10. Thus, in this case not only does the centroid not lie at the midpoint of the line connecting the inlet and outlet edge midpoints, it does not even lie on this line, eliminating the possibility for some sort of weighted DD scheme in which $\psi_{n,s} = \gamma\,\psi_{n,s,\mathrm{in}} + (1-\gamma)\,\psi_{n,s,\mathrm{out}}$, where $0 \leq \gamma \leq 1$, to restore second order convergence.

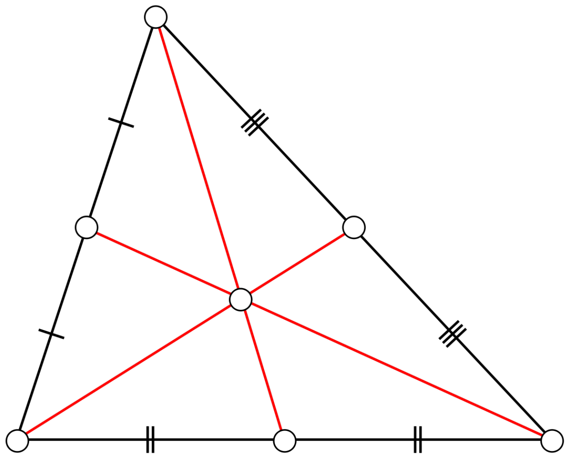


Figure 5.10: Centroid location of a triangle.

Having explained why the simple DD closure to the SBA framework is not second order convergent, Grove's initial hope that one dimensional schemes could work in the three dimensional SBA framework was not entirely wrong. As we have seen in previous sections, the SBA and ESBA using the DD scheme do indeed converge to the true solution, even if the convergence rate of the one dimensional scheme did not translate to the three dimensional SBA framework. A fruitful line of future research might be to seek a simple modification to the DD closure relation that will cause the modified SBA-DD and ESBA-DD methods to achieve second-order truncation error. We explored two ideas for this, without success.

Both attempts were characteristic in nature, attempting to find the average flux on the outlet face of the slice given the average flux on the inlet face of the slice, and then to use this outlet flux in the balance equation to determine the slice average flux. The first attempt defined the optical depth $\langle\tau\rangle$ between the inlet face centroid $\mathbf{r}_{c,\text{in}}$, and outlet face centroid $\mathbf{r}_{c,\text{out}}$

$$\langle\tau\rangle = \sigma_t \left|\mathbf{r}_{c,\text{in}} - \mathbf{r}_{c,\text{out}}\right| \, . \tag{5.23}$$

This optical depth was then used in the characteristic solution along the line connecting the inlet and outlet face centroids, after inserting the Padé (1,1) approximation for the resulting exponential terms

$$\psi_{n,s,\text{out}} = \psi_{n,s,\text{in}} \left(\frac{2-\langle\tau\rangle}{2+\langle\tau\rangle}\right) + \frac{q_{n,s}}{\sigma_t}\left(1 - \left(\frac{2-\langle\tau\rangle}{2+\langle\tau\rangle}\right)\right) \, . \tag{5.24}$$

The second attempt was slightly more complicated, and attempted to solve for the average flux on the outlet face as a linear combination of the characteristic solutions along the line connecting each pair of corresponding vertices of the inlet and outlet faces

$$\psi_{n,s,\text{out}} = \sum_{j=1}^{N_v} \beta_j \, \psi^j_{n,s,\text{out}} \, , \tag{5.25}$$

where N_v is the number of vertices on the inlet or outlet face, and $\psi^j_{n,s,\text{out}}$ is calculated using equation 5.24 after substituting τ_j for $\langle\tau\rangle$, which is defined similarly to $\langle\tau\rangle$

$$\tau_j = \sigma_t \left|\mathbf{r}_{j,\text{in}} - \mathbf{r}_{j,\text{out}}\right| \, . \tag{5.26}$$

The only loose end is then how to come up with a suitable set of $\{\beta_j\}_{j=1}^{N_v}$. This is done by determining what linear combination of the face vertices leads to the face centroid, and the result is

$$\beta_j = \frac{A_{j-1/2} + A_{j+1/2}}{2A_{\text{out}}} \ , \tag{5.27}$$

where $A_{j-1/2}$ is the area of the triangle formed by $\mathbf{r}_j$, $\mathbf{r}_{j-1}$, and $\mathbf{r}_{c,\text{out}}$, and $A_{j+1/2}$ is the area of the triangle formed by $\mathbf{r}_j$, $\mathbf{r}_{j+1}$, and $\mathbf{r}_{c,\text{out}}$. Again, it should be noted that both of these attempts resulted in first order convergence, and are only included here for the sake of anyone attempting a similar fix in the future.

5.2 Scaling

Parallel performance can be measured in at least two ways based on the application under consideration. One could imagine a problem that can fit into the memory of a single computer relatively easily, but is so computationally complex that it takes a long time to reach a satisfactory answer. In this case, super-computers and clusters are useful because more computational resources can be used to bring the time required to arrive at the solution down to a reasonable value. The second application type is one which is memory-bound rather than compute-bound. In this case, the problems of interest require so much memory that a single computer would not be sufficient. In this case, super-computers and clusters are useful because the problem can be subdivided into manageable portions such that each node of the machine is assigned a part of the problem that can fit into that node's memory. High fidelity transport calculations fit firmly into the second category due to the high dimensionality of the phase space.

To measure the parallel performance of a memory-bound application, a weak scaling study is typically performed. In this type of computational experiment, the size of the problem is kept proportional to the number of processing elements being used to solve it. For instance, if one node of a super-computer was used to solve a problem of a given size, eight nodes of the same machine would then be used to solve a similar problem with eight times the number of unknowns. The similar problem could be the same as the original problem but with a higher spatial resolution leading to eight times the number of spatial cells, or a higher angular resolution with eight times the number of angles in the S_N angular quadrature set, etc. The time taken to solve each problem is recorded as t_N, where N is the number processing elements used. For each data point in the study, the parallel efficiency is the ratio of t_1 to t_N. It is important to note that this ratio will be less than unity due to the communication time for distributed memory machines.

The weak scaling study performed in this research can be described as follows. For each core used, the resolution of the mesh is increased such that each core contains a subset of the global mesh with 1,000 cells. These 1,000 cells are arranged in a brick grid with ten divisions in each Cartesian dimension. The angular quadrature set is a S_{20} Gauss-Chebyshev angular quadrature consisting of 800 angles in total, and is kept constant throughout the study, as is the energy discretization with 12 groups. The group-to-group scattering scheme is Jacobi, so that all 12 groups are swept simultaneously. The domain is cube-shaped, and homogeneously filled with hydrogen at 10^{24} atoms/cm^3 and a unit strength volumetric source of particles in all 12 energy groups. The inner iteration scheme is source iteration, although full convergence of the inner iterations was not needed for this study of parallel sweeping. This is because the time measured in the scaling study is the time required for a single inner iteration, and therefore the number of iterations is irrelevant except for statistical variations in the individual data points. Each recorded time is taken to be the average of four inner iteration durations. Finally, the spatial discretization used in this study is the LDFE scheme. The results using the ESBA with parallelization option 1, alongside traditional transport sweeps for the SBA and CBA, are shown in Figure 5.11. The data points in this figure occur at 1, 8, 64, and 512 cores.

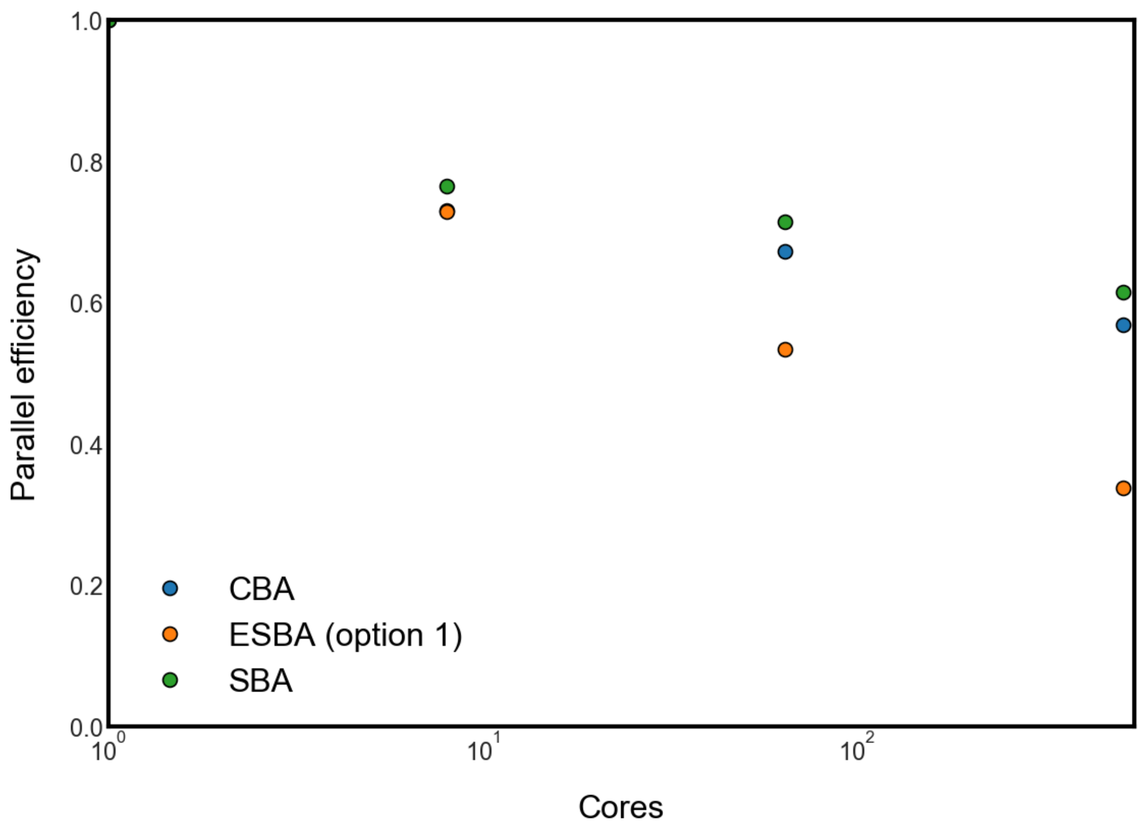


Figure 5.11: Weak scaling study results for parallelization option 1.

The results in Figure 5.11 are admittedly less than stellar, but the basic trend in which the SBA shows higher parallel efficiency than the CBA, which shows higher parallel efficiency than the ESBA with parallelization option 1, can be easily explained. We begin by noting that it is the computation to communication ratio which predominantly effects the parallel efficiency of this application. Thus, in order to increase the parallel efficiency, the biggest gains will be achieved by reducing the time spent communicating, or increasing the amount of work being done without also increasing the amount of communication required.

The difference in parallel efficiency between the CBA and SBA can thus be explained by examining how much work and how much communication is required by each. We begin by noting that in both the CBA and SBA, the same type of information is being communicated between nodes. For each face shared by two neighboring nodes, a data structure containing the facial flux moments on that face (four double precision floating point numbers), the global index of the face (one integer), and the energy group index (one integer), must be communicated to the neighboring node. This equates to 40 bytes per face assuming eight bytes per double precision floating point number and four bytes per integer. With each set of faces shared by neighboring nodes consisting of 100 faces, and all 12 groups being communicated together, each message contains $100 \times 12 \times 40$ B $= 48$ kB. This is true for both the CBA and SBA, and therefore the time spent communicating is expected to be equal between the two approaches. What changes between the two is the amount of work performed for each cell in the local mesh. This amount of work is much higher for the SBA than the CBA for the simple reason that the streaming plus collision operator is inverted on each slice rather than on each cell, but also because of the added work necessary to compute the necessary geometric information on each slice. This increase in work while keeping the communication time constant between the two approaches accounts for the higher parallel efficiency of the SBA relative to the CBA.

If we further consider the ESBA in relation to both the SBA and CBA, we note that the amount of work required by the ESBA is larger than for the SBA due to the addition of the sub-slice. While the streaming plus collision operator is still only being inverted on each slice, the work required to compute the geometric information on each sub-slice is not insignificant. The type and amount of information communicated by the ESBA using parallelization option 1 is quite different from the CBA and SBA. In the ESBA, we are no longer communicating angular flux moments on the faces shared by the two neighboring nodes, but the angular flux moments on the inlet facets of each slice residing on the faces shared by the neighboring nodes. For the Cartesian mesh considered in this study, and the angular quadrature set chosen, each boundary face will contain the inlet facets to three slices. Thus, without even considering the difference in the type of data structure that must be communicated, our communication costs have already tripled. As it turns out, the data structure being communicated by the ESBA with parallelization option 1 is only 4 bytes larger than the data structure used to communicate facial flux moments. This is because the face index alone is no longer sufficient to define a slice. We must also communicate the outlet face index on the neighboring processor, since any slice can be identified by an inlet outlet face pair. Thus, the data structure communicated in the ESBA is larger than that for the CBA and SBA by one integer value. The size of each message to be communicated is then $3 \times 100 \times 12 \times 44$ B = 158.4 kB. As can be gleaned from the data in Figure 5.11, this more than tripling of the communication cost significantly outweighs the modest increase in work, leading to lower parallel efficiencies for the ESBA with parallelization option 1.

Given these general trends, it should be noted that the parallel efficiencies shown in Figure 5.11 are much lower than expected for any of the three approaches. In particular, transport codes in existence today using the CBA with the LDFE spatial discretization scheme show parallel efficiencies of over 0.9 when considering cores counts in the thousands and even hundreds of thousands.[18] This less than ideal performance of the Slice-T code has been examined, and many changes to the code have produced negligible changes in these results. We leave the improvement of the parallel efficiency of all three approaches to future work, while reminding the reader that the Slice-T code is still in its infancy. Having said this, at the end of this section, we will still attempt to calculate the amount of time spent communicating using the specifications of the machine used to perform this study.

Next we consider a similar weak study using the ESBA with parallelization option 2. The results of this study are shown in Figure 5.12. Parallelization option 2 is more complex because given a number of cores, the problem can be solved with different numbers of angles per angle-set. For instance, consider the data point at 8 cores. Using parallelization option 2, we could solve this with a single angle per angle-set, in which case the spatial domain would be divided into 8 pipes, and each node would obtain the solution in a single pipe. If we were to use 2 angles per angle-set, the spatial domain would be split into 4 pipes for each of the two angles, resulting in 8 pipes in total for each of the eight cores to obtain the solution within. Finally, we could use 8 angles per angle-set, in which case the domain would remain undivided, and each of the eight cores would obtain the solution throughout the entire spatial domain for one of the 8 angles.

Parallelization option 2, yet again yields disappointing results, although the trend of higher parallel efficiencies for higher angle counts per angle-set can be seen clearly. To make sense of this, lets imagine the extreme case in which the number of angles per angle-set equals the total number of angles in the S_N angular quadrature set. Of course, this would require there to be as many cores as there are angles, so we will assume this. In this limit, each core must communicate the volumetric information of every single one of its local cells to every other core. This is because each core must sweep the entire spatial domain for a given angle. If we consider the case in which the number of angles per angle-set equals half the number of angles in the S_N angular quadrature set, the solution then proceeds in two steps, one for each angle-set. For each angle, the spatial domain is divided into two pipes, and the local cells residing along the division plane must be communicated to two other cores instead of just one as before. This increases the communication cost, leading to lower parallel efficiencies.

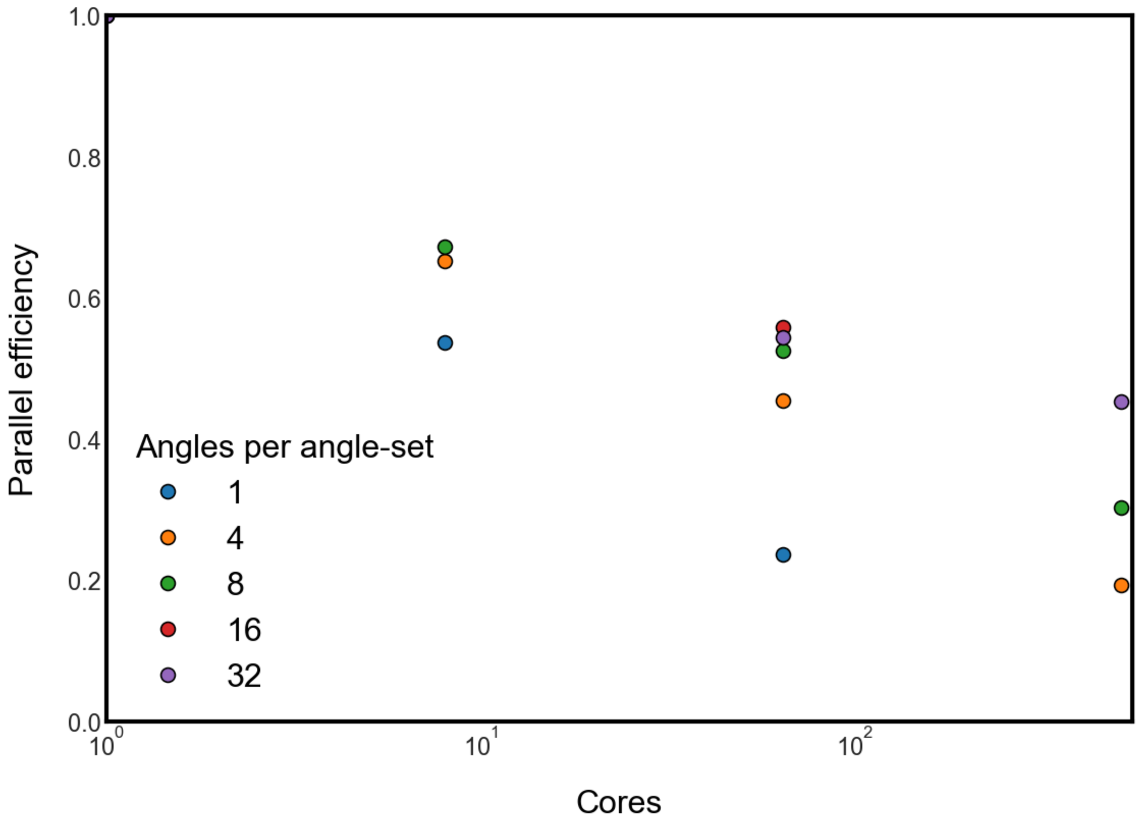


Figure 5.12: Weak scaling study results for parallelization option 2.

In an attempt to understand the low parallel efficiencies of the two weak scaling studies presented above, we will now attempt to estimate the theoretical communication time for the CBA with 8 cores, and compare this to the actual inner iteration durations recorded to determine exactly how far from optimal the performance actually is. On the machine used to perform these studies, the message passing latency is reported to be 1 μs, while the bandwidth is reported to be 20 GB/s. Using the 48 kB message size for the CBA, this equates to 0.002401 s per message sent. In the case of the 8 core run, there are 12 such messages occurring for each angle, with 800 angles in total. This leads to a total communication time of 0.033 s. For reference, the measured inner iteration duration for this case was 24.44 s. Thus, if the communication costs in the research code were as small as the latency and bandwidth suggest they could be with an optimal implementation, these costs would be a tiny fraction of the sweep time and efficiency would be close to unity. We leave such implementation improvements in Slice-T code to be pursued as future work.

5.3 GPU Acceleration

Before presenting the results of accelerating the slice and sub-slice formation process, we should more closely examine the GPU architecture and the specific GPUs used in this study, in order to gain a better understanding of how these devices work. In an attempt to explain such an architecture to the author, someone once posed the question "if you had to plough a field, would you use one ox, or a thousand chickens?" After a few years of experience and contemplation, the author has been convinced that the answer depends on the field, as previously discussed, but it also depends in large part on the chickens. How fast are they? How smart are they? Can they communicate with one another? As we will see, the answers to these questions are slow, stupid, and maybe, but there are a thousand of them.

The modern GPU is a collection of devices which NVIDIA has named streaming multi-processors (SMs). These SMs are different from the lowest level processing elements performing the computations, sometimes referred to as CUDA cores. A comparison is often made between the cores of a typical CPU and the processing elements of a GPU, which can number in the thousands. This analogy is flawed in the sense that each core of a CPU is much faster than each processing element of a GPU, each CPU core has access to its own cache, or set of caches, and each CPU core can work independently of the other cores on the CPU as long as care is taken to maintain cache coherency and avoid race conditions. The processing elements of a GPU operate much differently, and the better comparison to the CPU core is the

SM, not the individual processing elements.

Each processing element of the GPU works on a thread that executes the same kernel function with a different thread index. The work sent to the GPU is therefore a collection of threads, and these threads are organized into thread blocks, where each thread block is allocated to an SM. Within each thread block, the threads are further grouped into warps of 32 threads. The warps execute each instruction in the kernel function simultaneously. For example, if within the kernel function there is a line `a = b + 1`, all 32 threads will execute this instruction at the same time. This presents issues for branching statements, which should be kept to a minimum. For instance, if there is an `if-else` statement where half of the threads branch to `a = b + 1` and the other half branch to `a = b + 2`, these two instructions will be performed serially with half of the threads in the warp executing the first branch simultaneously, and only after these threads finish will the other threads execute the second branch simultaneously.

Complications for communication between threads arise from the different memory spaces that are available. The first of these is global memory which is typically between 10 and 20 GB, and is analogous to the RAM on the motherboard of the CPU. This global memory is one mechanism that threads can use to communicate with each other, but it is also quite slow to access (typically a bandwidth of a few hundred GB/s). For this reason, programmers are encouraged to make use of the shared memory of each SM, which can be accessed much faster (typically a bandwidth of 1-2 TB/s). Shared memory is essentially just a section of the L1 cache on each SM that the user can control, and it is yet another mechanism for communication between threads, but only for threads on the same SM, or alternatively between threads of the same thread block. To complicate things further, if synchronization is important, barriers can only be enforced between threads of the same thread block, which all reside on the same SM.

The end result is that the work to be performed on the GPU should be organized into thread blocks which are completely independent of each other, and there should be at least as many thread blocks as there are SMs on the GPU, but preferably many more. It is this coarse level of parallelism that achieves the first chunk of the speedup of any application. For instance, for thread blocks of the same size, running any number of thread blocks less than or equal to the number of SMs on the GPU would finish in roughly the same amount of time. Running one more thread block than the number of SMs on the device, on the other hand would take twice as much time, as would running twice as many thread blocks as there are SMs on the device. Thus, the number of thread blocks should be a multiple of the number of SMs if possible.

Figures 5.13 through 5.18 are diagrams of three GPUs closely resembling the GPUs used in this study, namely the NVIDIA K40, P100, and V100. The letters K, P, and V, stand for Kepler, Pascal, and Volta, and these names represent the "generation" of NVIDIA GPU technology. The difference between GPUs of different generations lies almost entirely in the SMs, and hence Figures 5.14, 5.16, and 5.18 examine the SMs of each of the three GPUs in detail. Different GPUs of the same generation mostly differ only in the number of SMs they contain, the amount of global memory they contain, and how they are connected to the host CPU, among other minor differences. Figures 5.13, 5.15, and 5.17 show the layouts of the GK110, GP100, and GV100, of which the K40, P100, and V100 respectively, are minor variations of. For instance, the P100 has only 56 SMs compared to the 60 shown in Figure 5.15.

Examining Figures 5.13 through 5.18, one trend is immediately obvious. This trend is the reduction in the number of processing elements per SM in favor of more

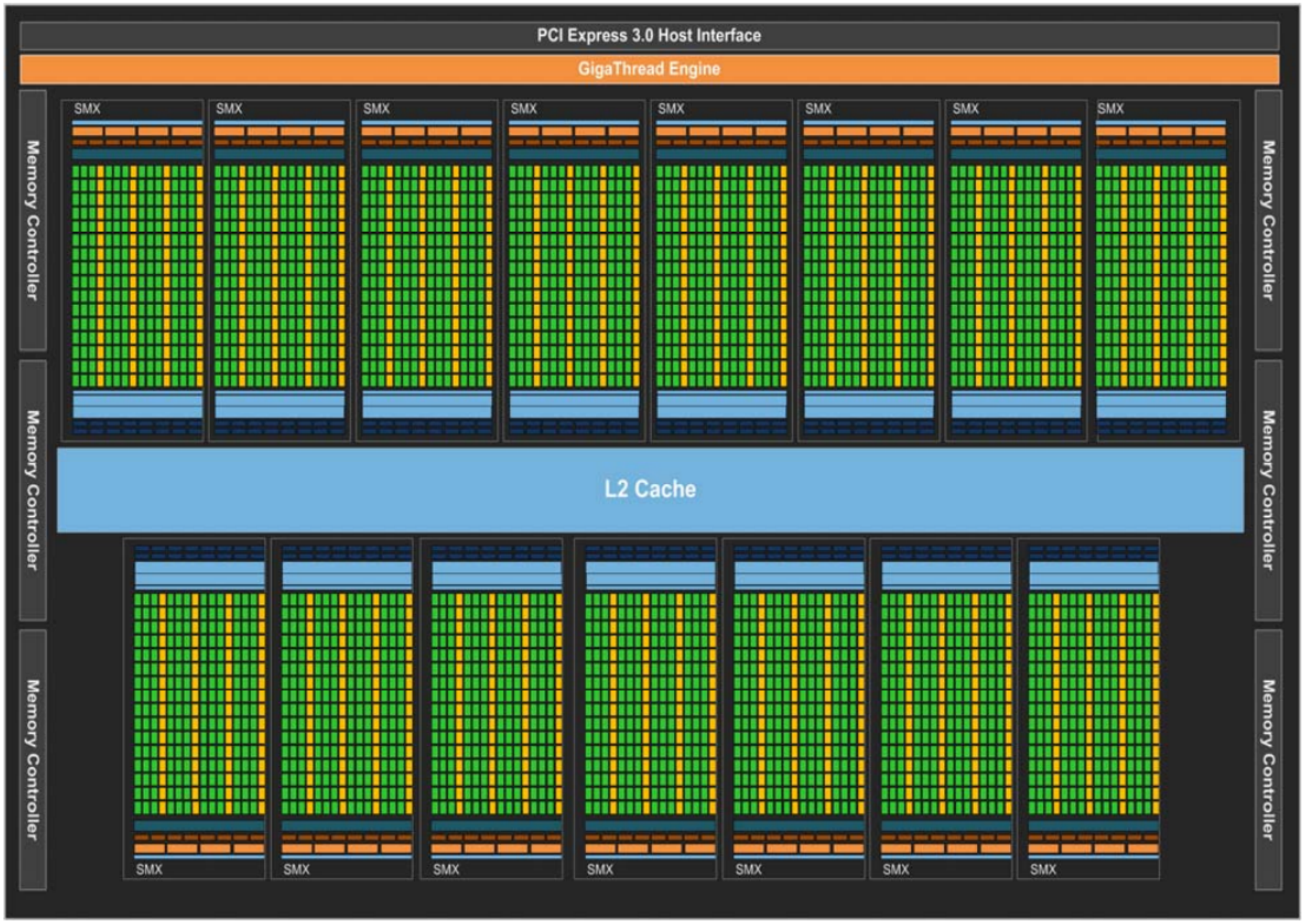


Figure 5.13: Streaming multiprocessor layout of the full GK110 GPU.[30]

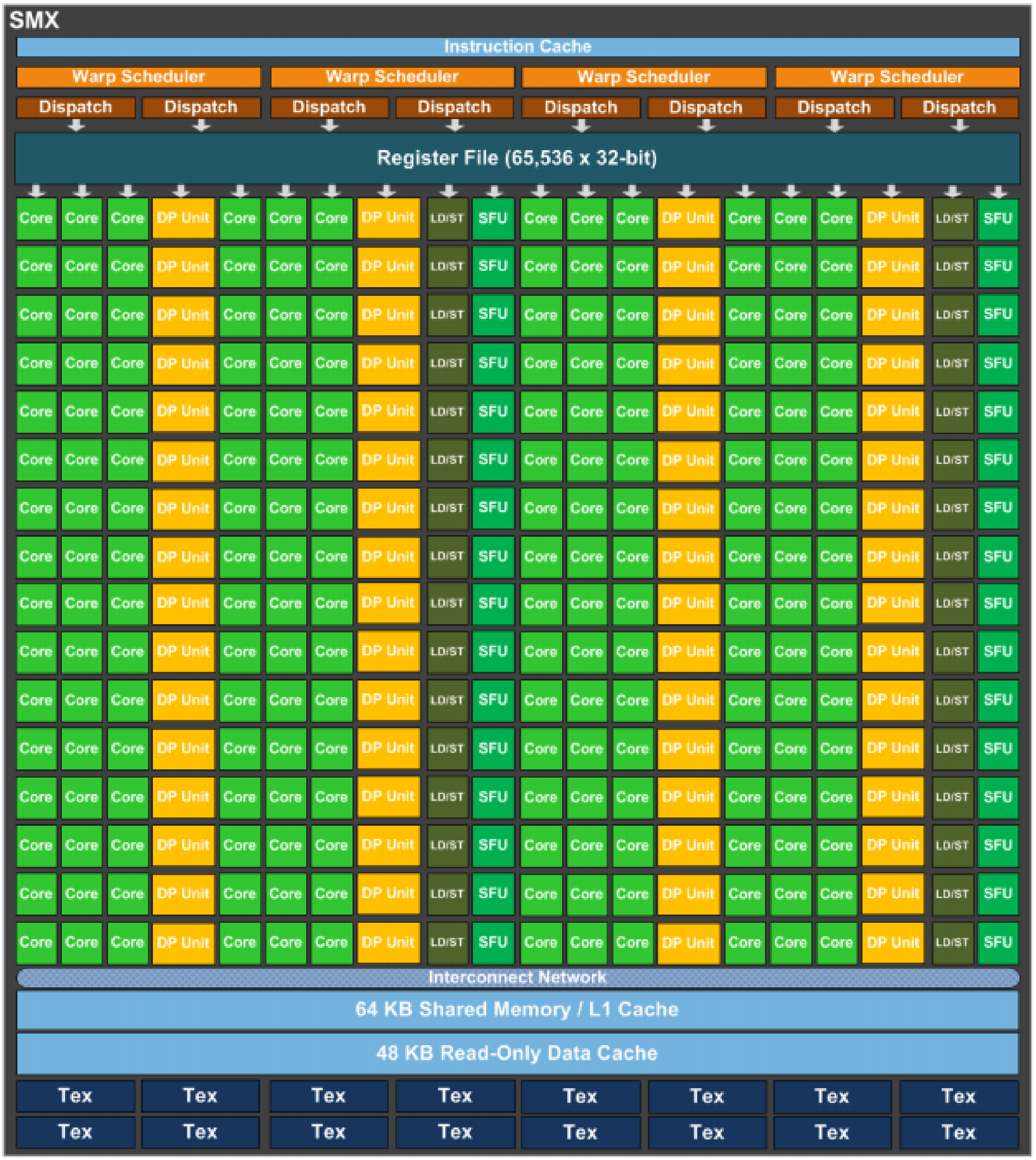

SMX
Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit)
Core
DP Unit
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
48 KB Read-Only Data Cache
Tex

Figure 5.14: Core layout of the GK110 streaming multiprocessor.[30]

SMs per GPU. In addition, while the ratio of double precision or floating point 64 (FP64) processing elements to single precision or floating point 32 (FP32) processing elements in the K40 was only 1:3, the same ratio increased to 1:2 for the P100 and

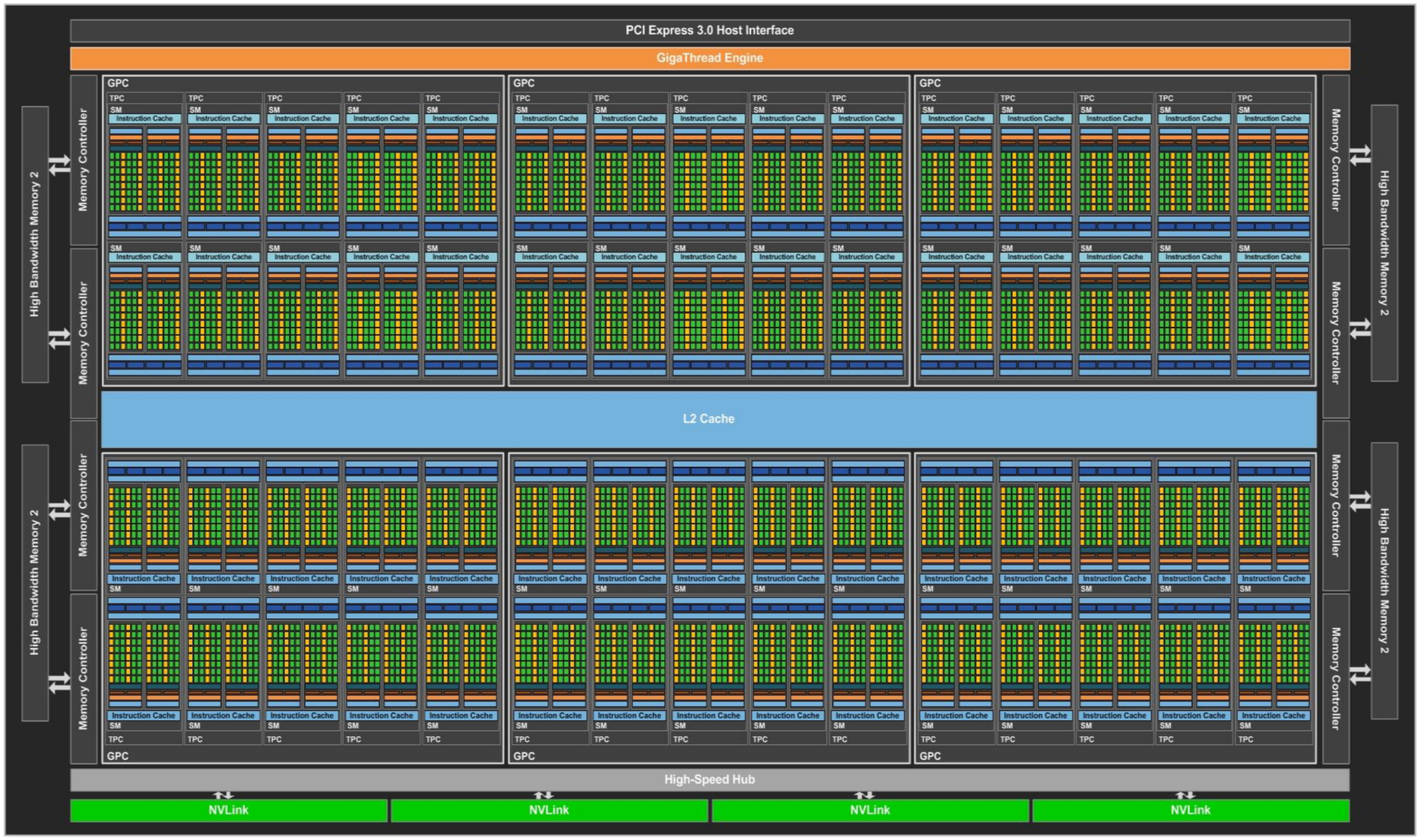


Figure 5.15: Streaming multiprocessor layout of the full GP100 GPU.[31]

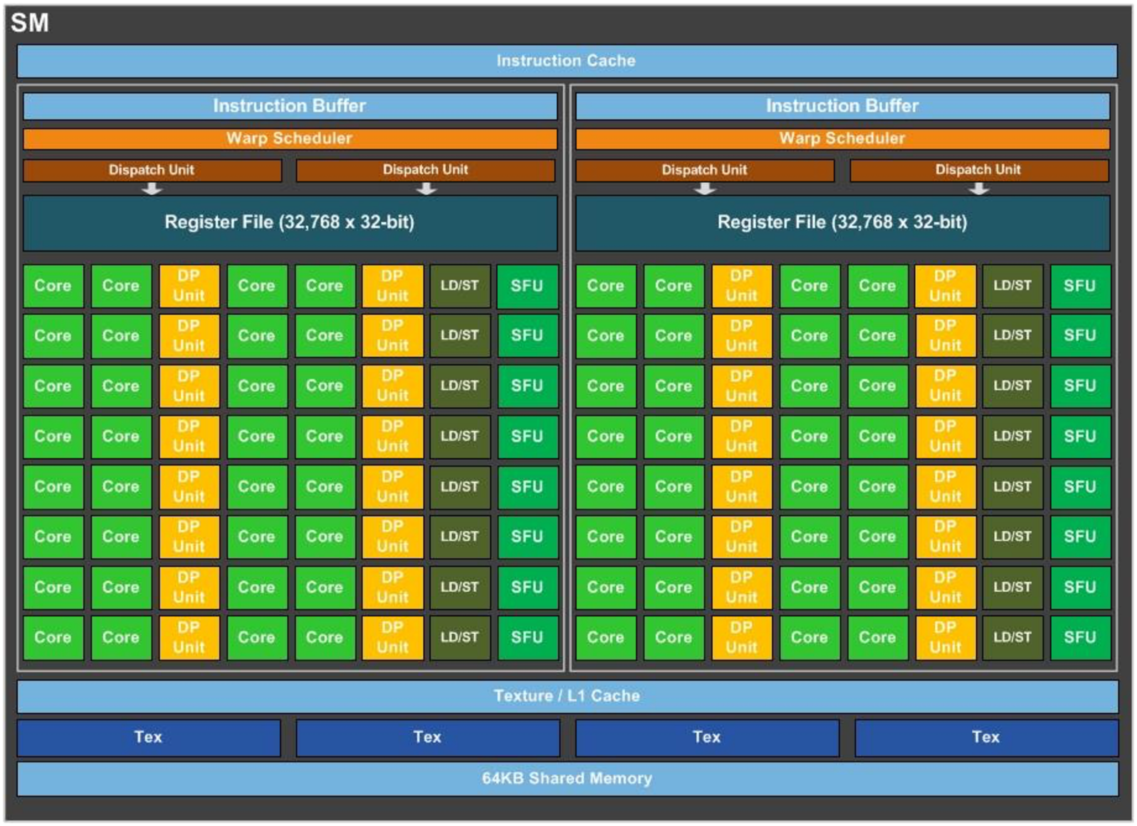


Figure 5.16: Core layout of the GP100 streaming multiprocessor.[31]

V100. This is particularly important for scientific computing where double precision arithmetic is the standard. Another trend is the heterogeneity of the SM cores themselves, utilizing different cores for different purposes such as the load-store (LD/ST) units, special function units (SFUs), and in the case of the V100, even cores specialized to handle integer (INT) and FP32 arithmetic, where these were handled by the same cores in previous generations. Also new in the V100 is the presence of the tensor cores, which is a fancy term for a vector processor optimized to handle mixed precision matrix arithmetic. While these tensor cores will not be utilized in this study, their effect on the theoretical peak performance is very intriguing. The specifications of the three GPUs used in this study are summarized in Table 5.2. As

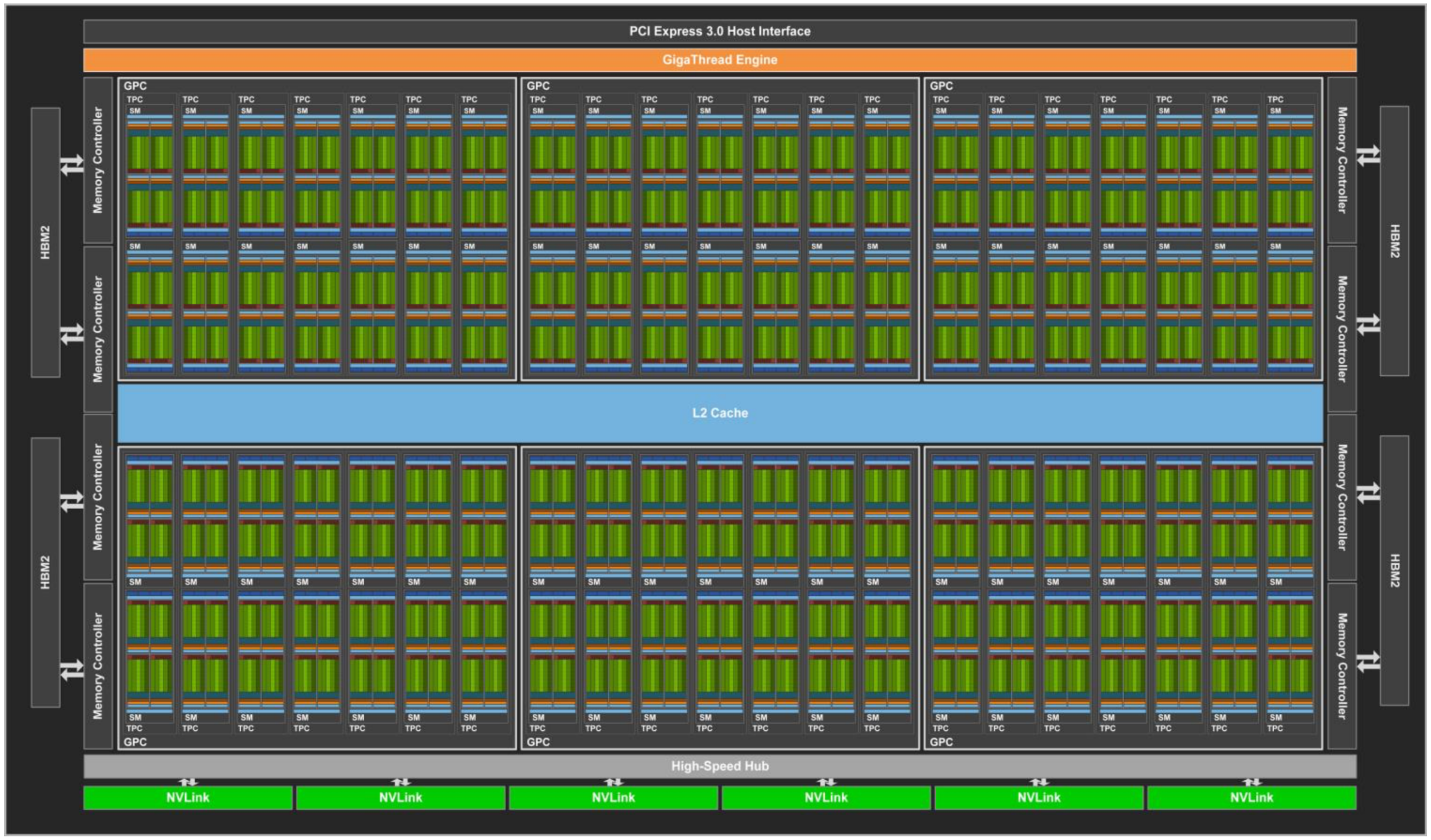


Figure 5.17: Streaming multiprocessor layout of the full GV100 GPU.[32]

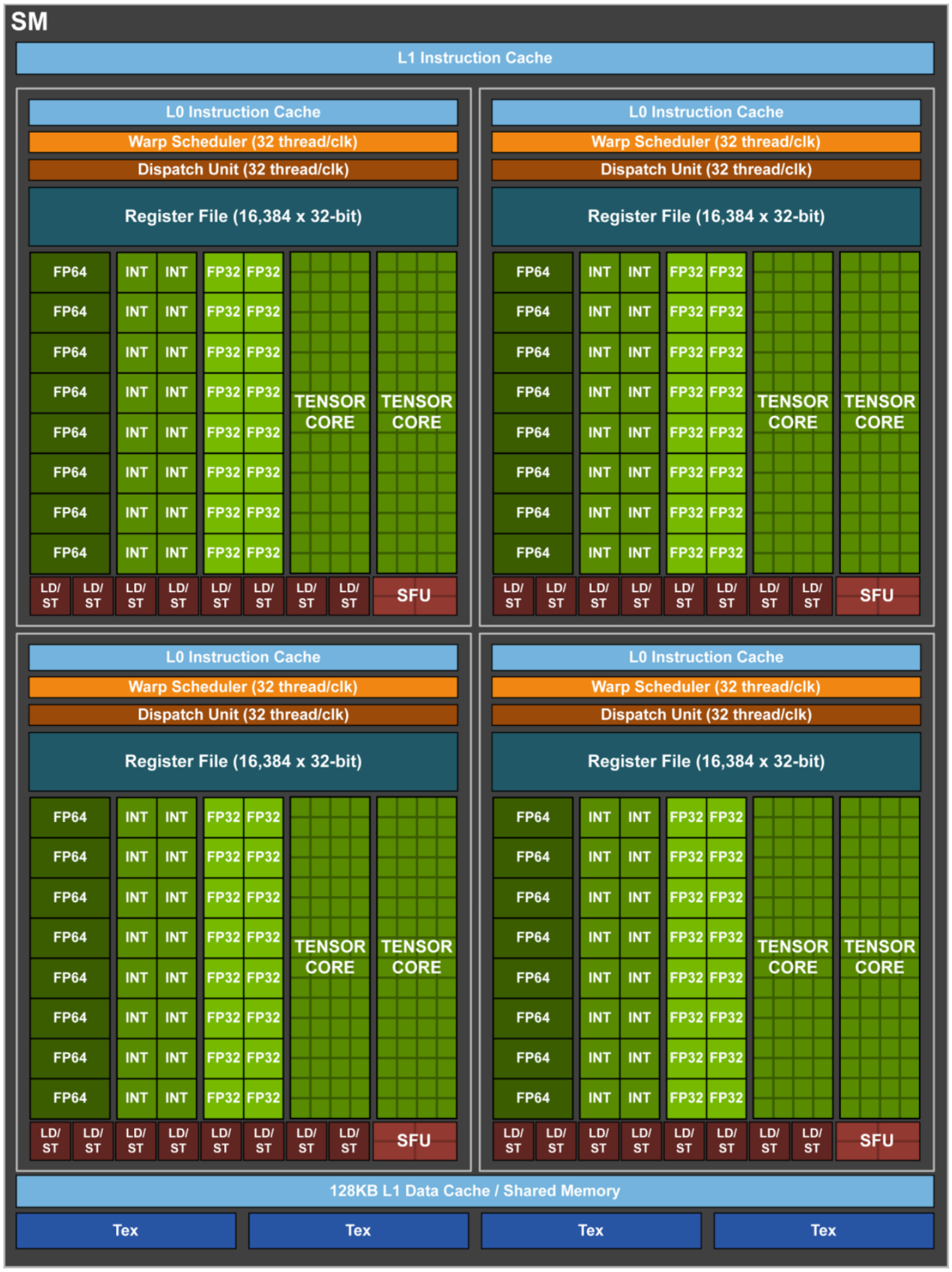

SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

Figure 5.18: Core layout of the GV100 streaming multiprocessor.[32]

Table 5.2: Comparison of the three GPUs used in this study.[30][31][32]

	K40	P100	V100
Number of SMs	15	56	80
FP32 Cores / SM	192	64	64
FP32 Cores / GPU	2880	3584	5120
FP64 Cores / SM	64	32	32
FP64 Cores / GPU	960	1792	2560
Max clock rate (MHz)	875	1480	1530
Memory (GB)	12	16	16
L2 cache (kB)	1536	4096	6144
Thermal design power (W)	235	300	300
Peak FP32 performance (TFLOPS)	5.0	10.6	15.7
Peak FP64 performance (TFLOPS)	1.7	5.3	7.8

can be seen in this table, the difference between the P100 and V100 is not quite as drastic as the difference between the K40 and the P100, however the increased core count and clock rate of the V100 compared to the P100 while maintaining the same thermal design power is truly amazing.

As mentioned in the previous chapter, four functions utilized in the local sweep to build the slices and sub-slices have been targeted for GPU acceleration. These four functions are `count_slices`, `slice_integration`, `count_sub-slices`, and `sub-slice_integration`. GPU kernel functions for these were written and optimized in terms of the number of threads per thread block and the amount of local memory used, and the results are shown in Figure 5.19. Each reported time is the average of 20 runs. This figure shows the run time for each function stacked on top of one another, and compares the GPU run times to the host CPU run times. In all three cases, the host CPU was an IBM POWER processor. The last case in the figure utilizing the IBM POWER9 and NVIDIA V100 is actually the exact combination that the Summit and Sierra systems feature.

The exact speedups between GPU and host CPU for the four functions targeted for GPU acceleration can be seen in Table 5.3. It can be seen that the speedups obtained over the host CPU increase from the K40 to the P100, and again from the P100 to the V100. Also, the speedups obtained for the counting functions are significantly higher than for the integration functions. In any case, Figure 5.19 shows that the time required for the slice and sub-slice preparation can be made vanishingly small compared to the actual local sweep, which was the desired outcome that could make the LDFE discretization viable within the SBA and ESBA frameworks. The dramatic speedups obtained are a direct result of the single instruction multiple data (SIMD) nature of the slice and sub-slice preparation process. Each slice and sub-slice can be worked on by a single thread, and no thread requires communication with any other thread. This is the type of application for which these devices are known to excel.

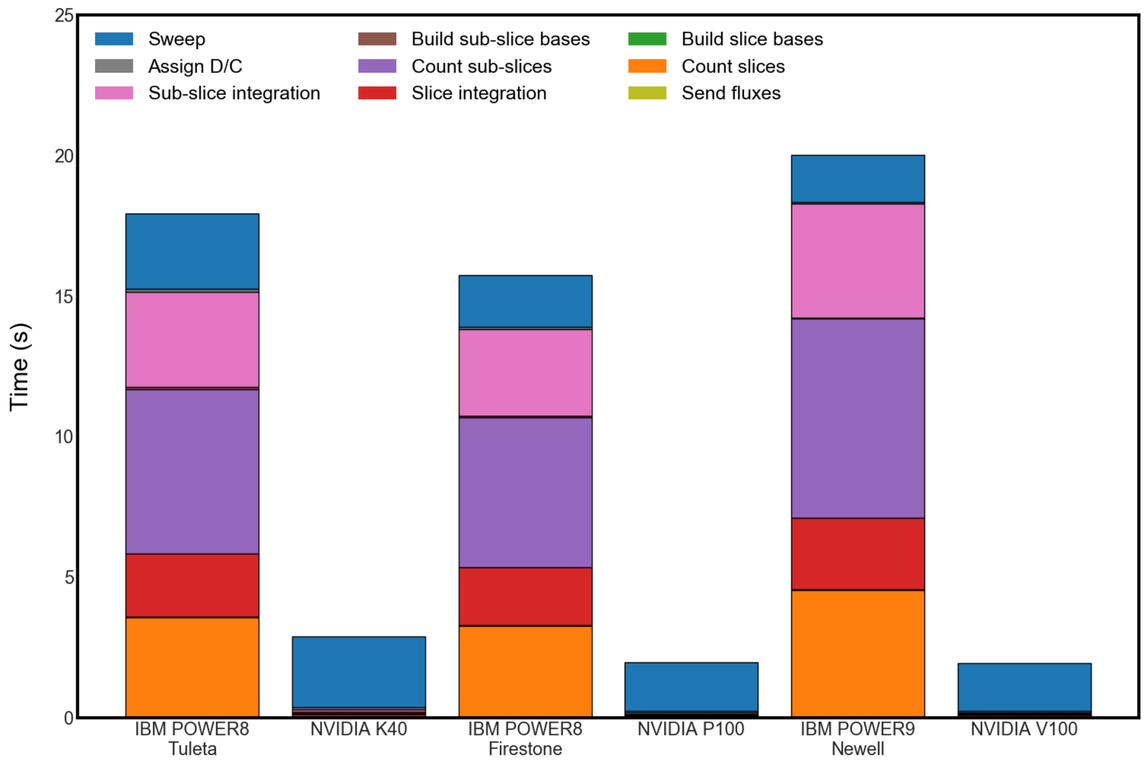


Figure 5.19: Comparison of the run times for each function in the local sweep for three CPUs and three GPUs.

Table 5.3: Speedups over host CPU for each GPU in the study and for each function targeted for GPU acceleration.

	K40	P100	V100
`count_slices`	214.4	313.5	362.9
`slice_integration`	25.1	43.0	44.1
`count_sub-slices`	223.0	330.7	400.9
`sub-slice_integration`	38.6	88.7	130.8

5.4 Application

No description of a code or numerical method would be complete without a demonstration of its abilities to solve actual problems of interest as opposed to simple manufactured solutions. To this end, two such problems are chosen to demonstrate the capabilities of the Slice-T code. The first problem is the steady state solution for a small localized source surrounded by a much larger volume of homogeneous dry air, using the DLC-31 unclassified thermonuclear neutron energy spectrum.[33] This problem was chosen due to one of the original motivations to write the Slice-T code, which was to generate transport libraries for the Neutron Gamma Energy Transport (NuGET)[34] fratricide and hostile environment code developed at Sandia National Laboratories (SNL). NuGET uses precomputed air transport libraries and mass integral scaling to compute the survivability of nuclear weapon components.

The second problem of interest is chosen to demonstrate the method's ability to handle complex geometries, and if we are being honest, because it is amusing. The problem is to solve for the scalar flux resulting from a one-group, steady state, volumetric particle source in the shape of my dog Jethro. This is the not-so-real-world problem referenced at the beginning of this chapter, and while there is no compelling application behind this problem, the complexity of the geometry is meant to test the algorithm for slice and sub-slice formation.

5.4.1 Unclassified Thermonuclear Point Source in Homogeneous Dry Air

The Slice-T code was used to compute the steady state neutron scalar flux solution due to a small localized source in homogeneous dry air. The neutron energy spectrum, given in Table 5.4 and plotted in Figure 5.20, was taken from the DLC-31 unclassified thermonuclear neutron energy spectrum,[33] and re-binned into the NuGET energy group structure. This group structure contains 89 groups, as well as all of the group boundaries contained in the DLC-31 group structure. In Figure 5.20, the peak at the upper end of the energy range represents the 14.1 MeV D-T fusion neutron source, while the broader spectrum represents neutrons occurring from fast fission. Table 5.5 gives the composition of dry air used in the simulation. All cross sections were obtained using NJOY with a uniform shape function, a temperature of 296 K, and a P_8 Legendre angular expansion. Raw data for these cross sections were retrieved from the ENDF/B-VII.1 library of incident-neutron data. Only elastic scattering events were considered due to the current limitations of the Slice-T code.

Figures 5.21 through 5.25 attempt to highlight certain aspects of the spatial mesh. The mesh boundary is a square prism spanning from -2,828.4271 cm to 2,828.4271 cm in the x and y dimensions, and -4,000.0000 cm to 4,000.0000 cm in the z dimension. Figure 5.21 shows the top face of the mesh looking down the z axis. This figure attempts to show how the mesh changes from Cartesian near the outer boundary, to cylindrical as we move towards the z axis. The radial bins then decrease in size logarithmically as we move closer to the z axis, as do the vertical bins as we approach the source location at $z = 0$ as shown in Figure 5.22, until once again the mesh becomes a Cartesian grid centered on the z axis with $\Delta x = \Delta y = 1$ cm, shown in Figure 5.25. Cells with centroids inside a sphere centered at the origin with radius $r = 5$ were deemed source cells resulting in the source shape shown in Figure 5.24.

Table 5.4: Tabulated values for the DLC-31 unclassified thermonuclear neutron energy spectrum.

Group number	Lower bound (eV)	Upper bound (eV)	$\phi(E)$ $\left(\frac{\text{n}}{\text{cm}^2\cdot\text{s}\cdot\text{eV}\cdot\text{steridian}}\right)$
1	1.7×10^7	2.0×10^7	0.0
2	1.5×10^7	1.7×10^7	0.0
3	1.3×10^7	1.5×10^7	2.47678576×10^5
4	1.2×10^7	1.3×10^7	2.57097795×10^5
5	1.0×10^7	1.2×10^7	1.30346444×10^5
6	7.8×10^6	1.0×10^7	8.03793852×10^4
7	6.1×10^6	7.8×10^6	1.02173042×10^5
8	4.7×10^6	6.1×10^6	1.32190668×10^5
9	3.7×10^6	4.7×10^6	2.10085941×10^5
10	2.9×10^6	3.7×10^6	2.70538896×10^5
11	2.2×10^6	2.9×10^6	4.09202272×10^5
12	1.7×10^6	2.2×10^6	5.98775729×10^5
13	1.4×10^6	1.7×10^6	8.96337237×10^5
14	1.2×10^6	1.4×10^6	8.95900691×10^5
15	1.1×10^6	1.2×10^6	1.16193799×10^6
16	9.3×10^5	1.1×10^6	1.58509204×10^6
17	8.2×10^5	9.3×10^5	1.58445146×10^6
18	7.2×10^5	8.2×10^5	1.58755106×10^6
19	6.4×10^5	7.2×10^5	1.58559563×10^6
20	5.6×10^5	6.4×10^5	1.58379792×10^6
21	5.0×10^5	5.6×10^5	2.17084326×10^6
22	4.4×10^5	5.0×10^5	2.32710427×10^6
23	3.9×10^5	4.4×10^5	2.32688941×10^6
24	3.0×10^5	3.9×10^5	2.32786289×10^6

25	2.4×10^5	3.0×10^5	2.32933373×10^6
26	1.8×10^5	2.4×10^5	2.33331027×10^6
27	1.4×10^5	1.8×10^5	2.39192103×10^6
28	1.1×10^5	1.4×10^5	2.49282536×10^6
29	8.7×10^4	1.1×10^5	1.89426240×10^7
30	6.7×10^4	8.7×10^4	1.89421308×10^7
31	5.2×10^4	6.7×10^4	1.89430086×10^7
32	4.1×10^4	5.2×10^4	1.94912897×10^7
33	3.2×10^4	4.1×10^4	1.95053429×10^7
34	2.8×10^4	3.2×10^4	1.95450187×10^7
35	2.6×10^4	2.8×10^4	1.95407241×10^7
36	2.5×10^4	2.6×10^4	1.95593780×10^7
37	2.2×10^4	2.5×10^4	1.95038517×10^7
38	1.9×10^4	2.2×10^4	8.02332944×10^7
39	1.7×10^4	1.9×10^4	8.02626543×10^7
40	1.5×10^4	1.7×10^4	8.02439560×10^7
41	1.3×10^4	1.5×10^4	8.02793318×10^7
42	1.2×10^4	1.3×10^4	8.02520205×10^7
43	1.0×10^4	1.2×10^4	8.02403343×10^7
44	9.1×10^3	1.0×10^4	1.66272690×10^8
45	8.0×10^3	9.1×10^3	1.66273915×10^8
46	7.1×10^3	8.0×10^3	1.66273689×10^8
47	6.3×10^3	7.1×10^3	1.66289430×10^8
48	5.5×10^3	6.3×10^3	1.66259742×10^8
49	4.9×10^3	5.5×10^3	1.66298400×10^8
50	4.3×10^3	4.9×10^3	1.66252249×10^8
51	3.8×10^3	4.3×10^3	1.66288639×10^8

52	3.4×10^3	3.8×10^3	1.66260600×10^8
53	3.0×10^3	3.4×10^3	3.48100457×10^8
54	2.6×10^3	3.0×10^3	3.48192064×10^8
55	2.3×10^3	2.6×10^3	3.48120521×10^8
56	2.0×10^3	2.3×10^3	3.48157807×10^8
57	1.8×10^3	2.0×10^3	3.48103011×10^8
58	1.6×10^3	1.8×10^3	3.48115545×10^8
59	1.4×10^3	1.6×10^3	3.48113212×10^8
60	1.2×10^3	1.4×10^3	3.48174559×10^8
61	1.1×10^3	1.2×10^3	3.56420690×10^8
62	9.6×10^2	1.1×10^3	3.56376465×10^8
63	7.5×10^2	9.6×10^2	3.56394450×10^8
64	5.8×10^2	7.5×10^2	3.56399408×10^8
65	4.5×10^2	5.8×10^2	4.22756650×10^8
66	3.5×10^2	4.5×10^2	4.22779825×10^8
67	2.8×10^2	3.5×10^2	4.22714523×10^8
68	1.7×10^2	2.8×10^2	4.21843363×10^8
69	1.0×10^2	1.7×10^2	4.21790323×10^8
70	6.1×10^1	1.0×10^2	2.62963771×10^8
71	3.7×10^1	6.1×10^1	2.62949454×10^8
72	2.3×10^1	3.7×10^1	1.47831890×10^8
73	1.4×10^1	2.3×10^1	0.0
74	8.3	1.4×10^1	0.0
75	5.0	8.3	0.0
76	3.1	5.0	0.0
77	1.1	3.1	0.0
78	8.0×10^{-1}	1.1	0.0

79	6.0×10^{-1}	8.0×10^{-1}	0.0
80	4.1×10^{-1}	6.0×10^{-1}	0.0
81	2.0×10^{-1}	4.1×10^{-1}	0.0
82	1.5×10^{-1}	2.0×10^{-1}	0.0
83	1.0×10^{-1}	1.5×10^{-1}	0.0
84	7.0×10^{-2}	1.0×10^{-1}	0.0
85	3.0×10^{-2}	7.0×10^{-2}	0.0
86	1.0×10^{-2}	3.0×10^{-2}	0.0
87	5.0×10^{-3}	1.0×10^{-2}	0.0
88	1.0×10^{-3}	5.0×10^{-3}	0.0
89	1.4×10^{-4}	1.0×10^{-3}	0.0

The solution was obtained using the ESBA-LDFE combination of balance approach and spatial discretization, source iteration for the inner iterations, Gauss-Seidel for the outer iterations. and a S_{20} Gauss-Chebyshev angular quadrature consisting of 800 angles in total. The mesh contained 887,040 hexahedral cells. A sample of the results obtained from the simulation is shown in Figures 5.26 through 5.32. The first three figures show the scalar flux solution in group 19 from three different viewpoints. The next two figures show the scalar flux in group 60 from two different viewpoints. The last two figures show the scalar flux in groups 76 and 89. Each of these figures show contour plots of the scalar flux at various representative values.

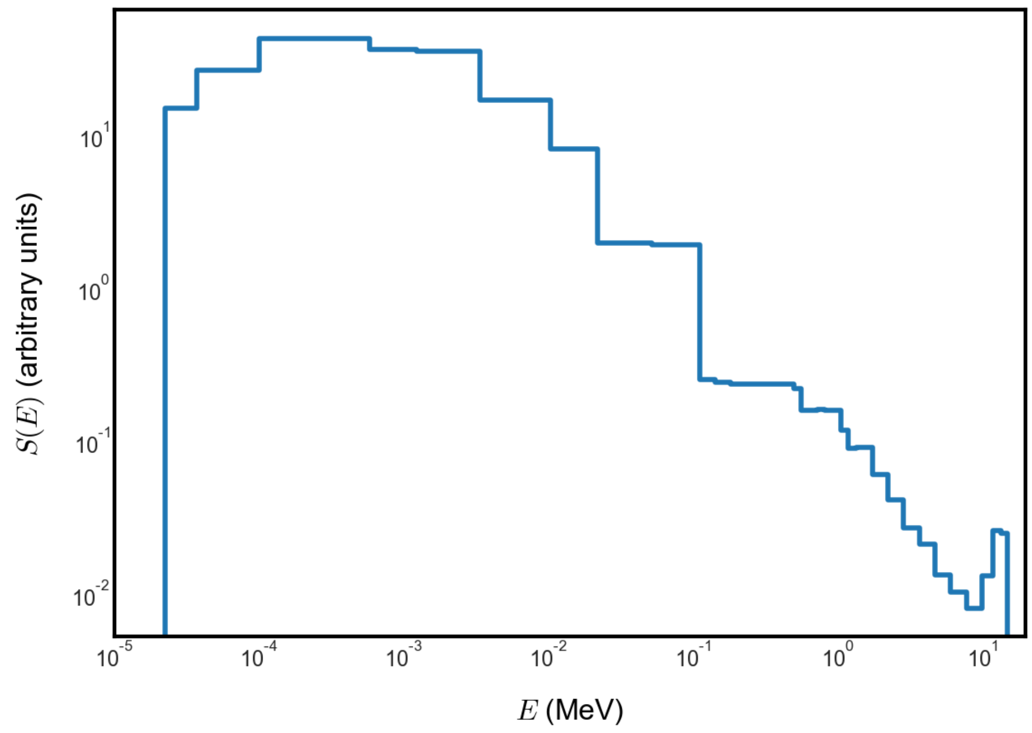


Figure 5.20: Plot of the DLC-31 unclassified thermonuclear neutron energy spectrum.

Table 5.5: Isotopic description of dry air used in the point source problem.

Material	Atomic density (atoms/cm^3)
C (natural abundances)	6.4024000×10^{15}
^{14}N	3.3138780×10^{19}
^{15}N	1.2105200×10^{17}
^{16}O	8.9140064×10^{18}
^{17}O	2.1708800×10^{16}
^{36}Ar	6.7840000×10^{14}
^{38}Ar	1.2720000×10^{14}
^{40}Ar	1.9728720×10^{17}

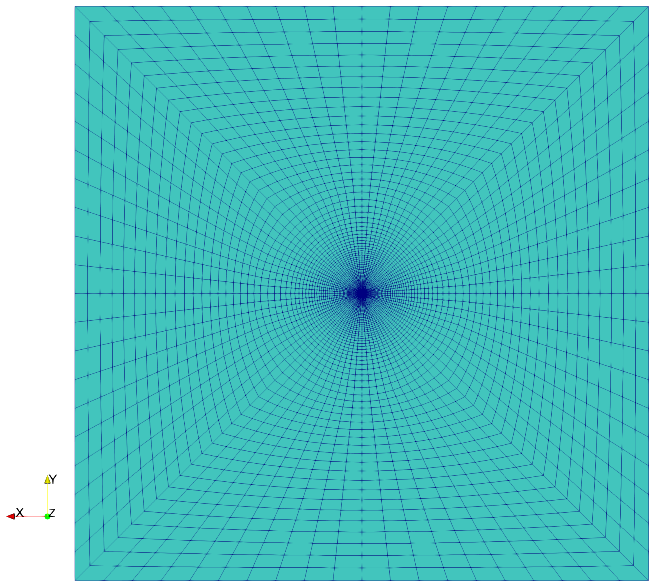


Figure 5.21: Top view of the mesh used in the point source problem.

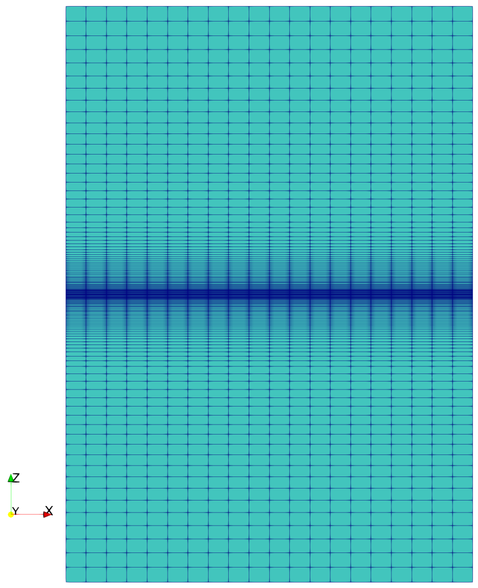


Figure 5.22: Side view of the mesh used in the point source problem.

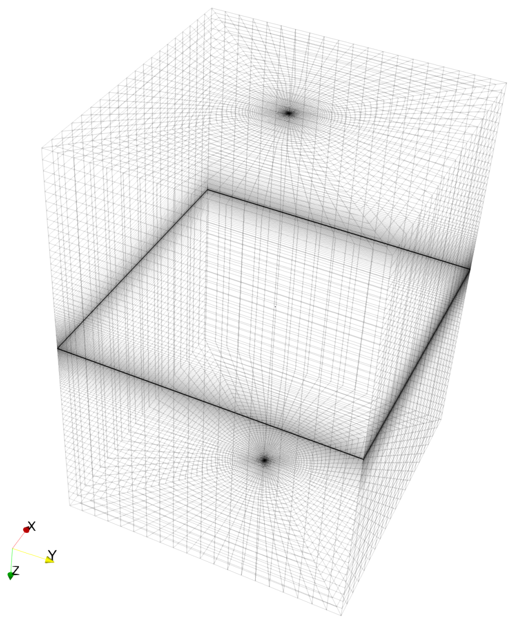


Figure 5.23: Tilted view of the mesh used in the point source problem.

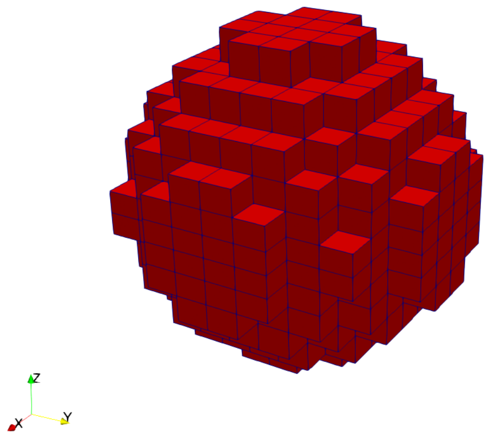


Figure 5.24: Close up view of the approximation to a point source used in the point source problem.

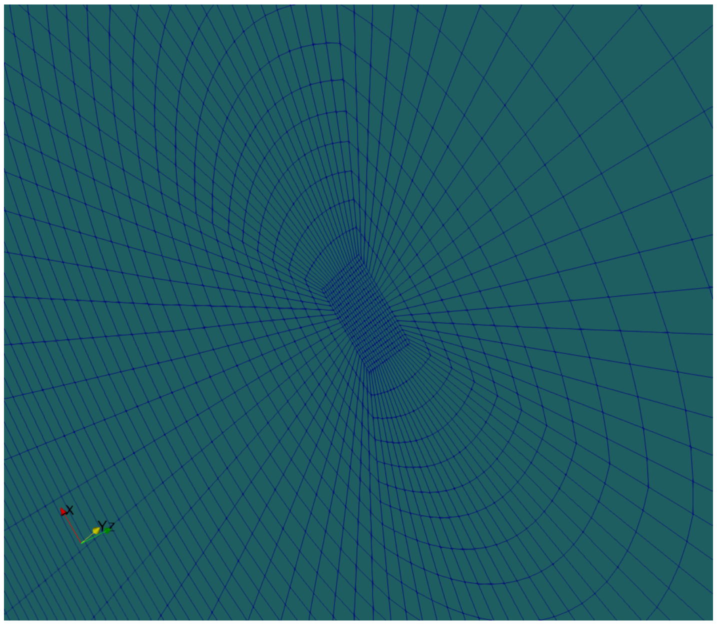


Figure 5.25: Close up view of the top of the mesh near the cylindrical axis for the mesh used in the point source problem.

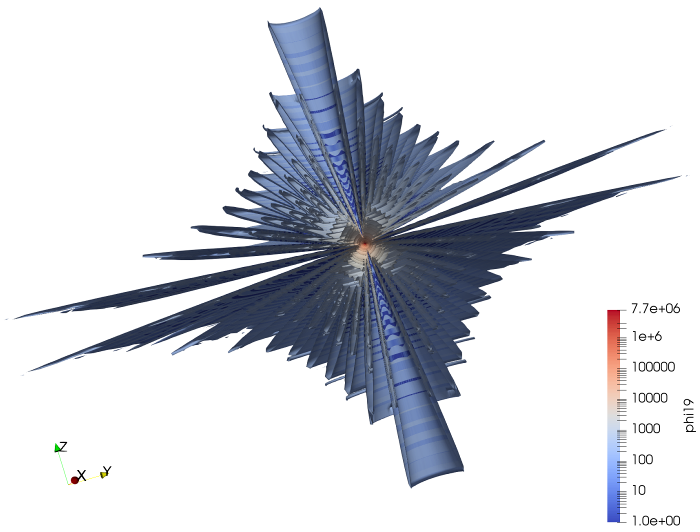


Figure 5.26: Cutaway view of the scalar flux contours for group 19.

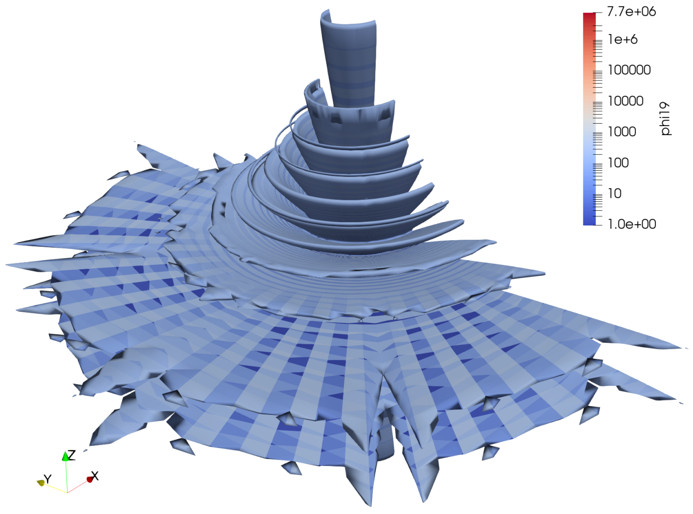


Figure 5.27: Rear view of the scalar flux contours for group 19.

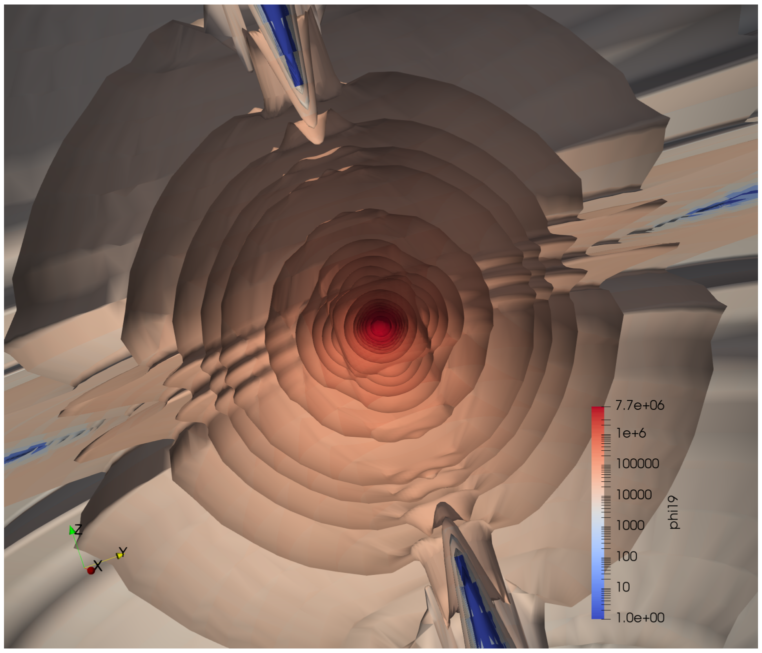


Figure 5.28: Zoomed in cutaway view of the scalar flux contours for group 19.

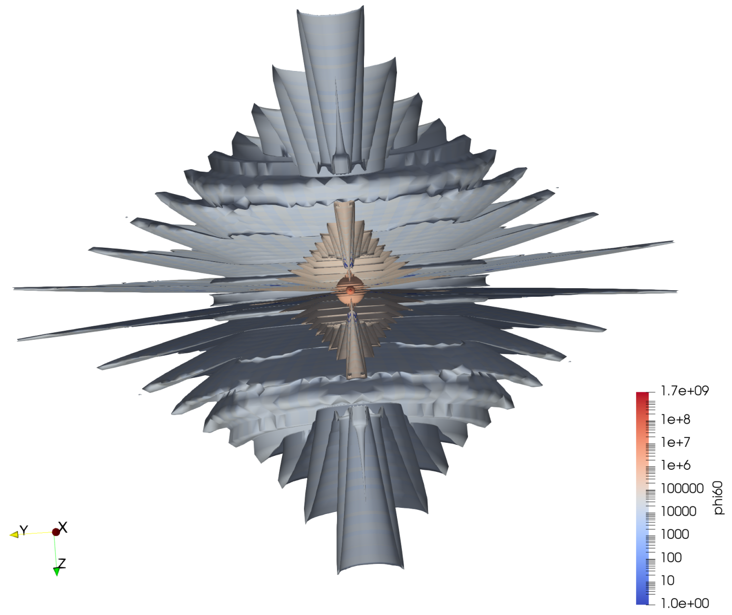


Figure 5.29: Cutaway view of the scalar flux contours for group 60.

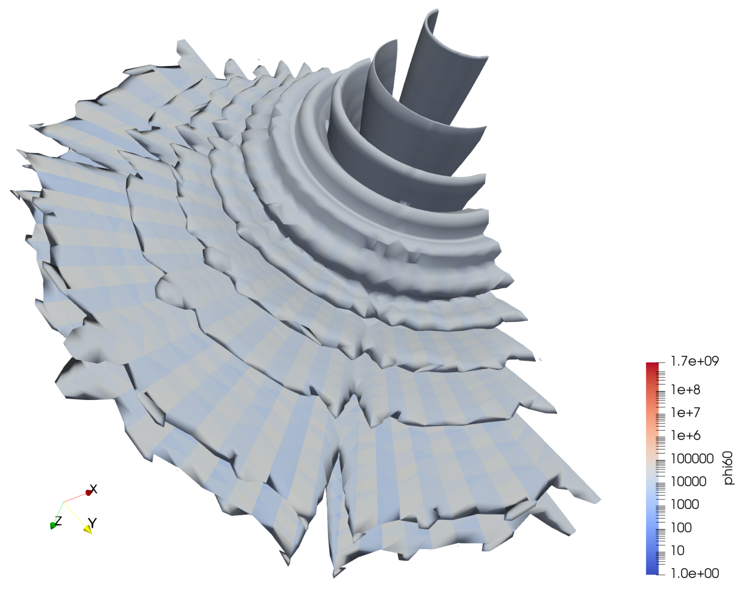


Figure 5.30: Rear view of the scalar flux contours for group 60.

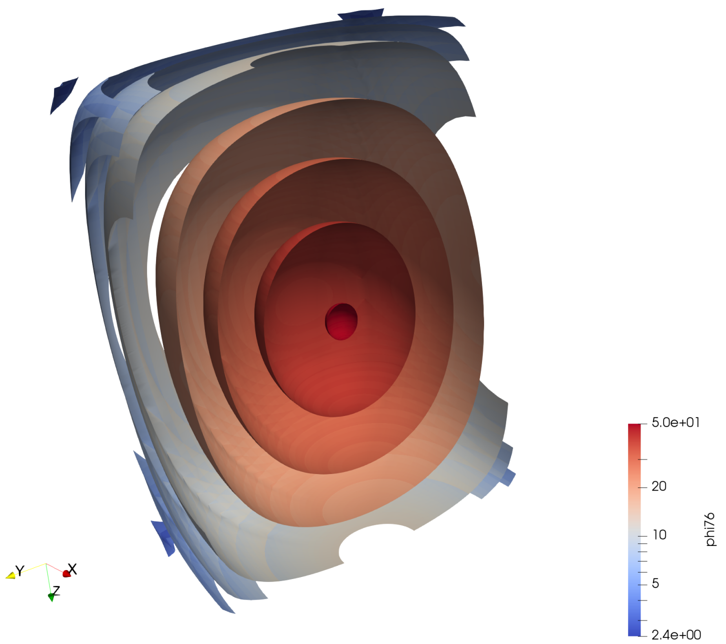


Figure 5.31: Cutaway view of the scalar flux contours for group 76.

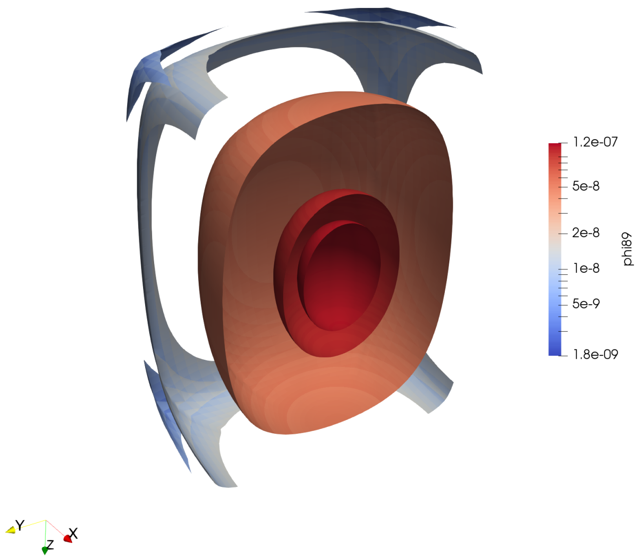


Figure 5.32: Cutaway view of the scalar flux contours for group 89.

The results from Figures 5.26 through 5.32 show several interesting characteristics, some of which were anticipated. The first such characteristic is the presence of ray effects in the higher energy groups from which source particles are emitted. This is a well known deficiency of discrete ordinate methods for which there exists some remedies, however none have been implemented into the Slice-T code to date. These ray effects are the result of the small localized source and the discrete nature with which the angular variable is treated via collocation. Accurate spatial discretization methods on fine spatial meshes will reveal ray effects in their full severity in problems like this. Coarse meshes or diffusive discretization methods will smear out the ray effects. The fine radial mesh here allows the ESBA-LD scheme to resolve the "rays" that correspond to discrete polar directions, but the coarse azimuthal mesh smears out the rays that correspond to discrete azimuthal directions. A less accurate method would smear out the polar "rays" even though the radial mesh is relatively fine.

As we examine the solution for lower energy groups for which no source particles are emitted, we begin to see a more reasonable looking solution, namely a smoothly decreasing scalar flux with distance from the localized source. The smooth nature of the solution is due to the fact that all particles in the higher energy groups arrived there via energy decreasing scattering reactions, and hence are influenced less by the location of the source as by the spatial distribution of the scalar flux in the higher energy groups. It is well known that scattering tends to decrease the severity of ray effects in discrete ordinates calculations, because the scattering source is distributed over a large volume instead of being localized. The elliptical shape of the contours in the last two images are nevertheless concerning, as they should in theory by perfectly spherical.

This elliptical stretching of the solution in the lower energy groups is almost certainly due to the cylindrical nature of the mesh, and in particular to the difference in radial and axial mesh spacing. With the benefit of hindsight, the mesh should have attempted to mimic the spherical nature of the solution rather than the cylindrical nature of the problem. While the problem as stated may not be cylindrical in nature, the ultimate goal of this type of simulation is to add an air-ground interface. After the introduction of this interface, the problem does indeed become cylindrically symmetric, and this was the reason for the design of the mesh.

5.4.2 The Jethro Problem

It is often the case that even when a numerical method or code has the ability to handle arbitrary polyhedral meshes, results are only presented on Cartesian or purely hexahedral meshes as has been done up to this point in this document. For this reason, and admittedly for fun, we now consider a volumetric, isotropic, unit strength radiation source in the shape of the my dog, Jethro. The simulation uses an S_4 quadrature set, consisting of 32 total angles, isotropic scattering, and a single energy group. Jethro is depicted in Figure 5.33, the geometric model used in the simulation is shown in Figure 5.34, and the scalar flux solution at one mid-plane of the cubic bounding domain is shown in Figure 5.35.

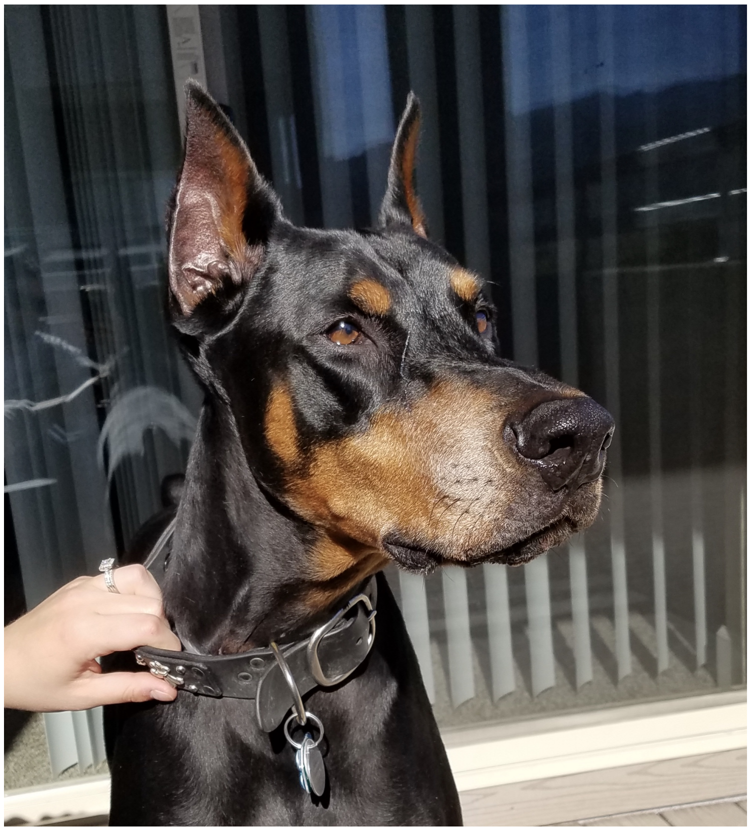


Figure 5.33: Jethro.

While it is obvious that such a problem has no analytical solution for which to compare, it is the author's hopes that such a complicated geometry could be used in the future to test other codes' abilities to handle complicated tetrahedral meshes in which the cell volume changes drastically over relatively short distances. One possible use for this mesh that is currently under discussion is the use of the Jethro Problem as a criticality safety benchmark. For instance, one could ask if Jethro were composed of solid plutonium, what density would be required to cause criticality? Various different codes would provide different answers, and reasons for these discrepancies could then be explored.

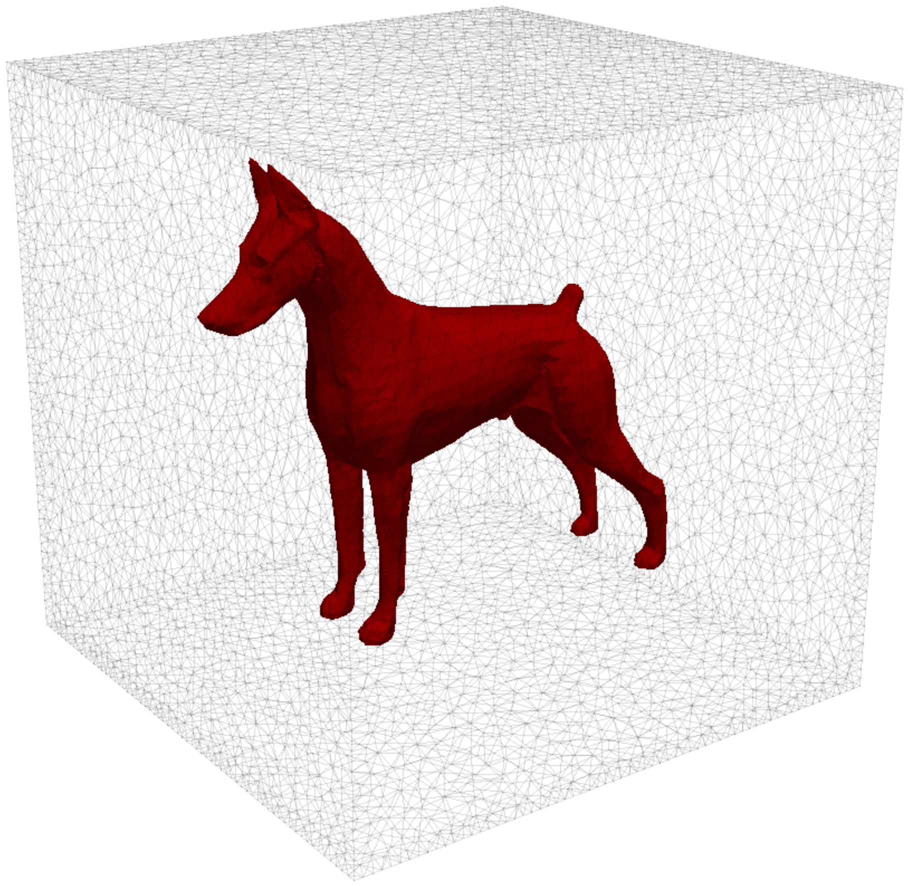


Figure 5.34: Jethro-shaped volumetric source.

Having acknowledged that no analytical solution exists, the numerical solution at the mid-plane seems reasonable. For instance, the large barrel-shaped chest region produces a significant bulge in the scalar flux as one would expect, and the scalar flux drops quickly just outside the surface of the source. Aside from qualitative observations such as these, complicated problems lead to relatively few conclusions about the accuracy of a code without experimental evidence, of which there is obviously none. However, the code was able to handle this complicated mesh with its highly varying mesh density, forming slices and sub-slices and executing slice-based sweeps, as it was designed to do.

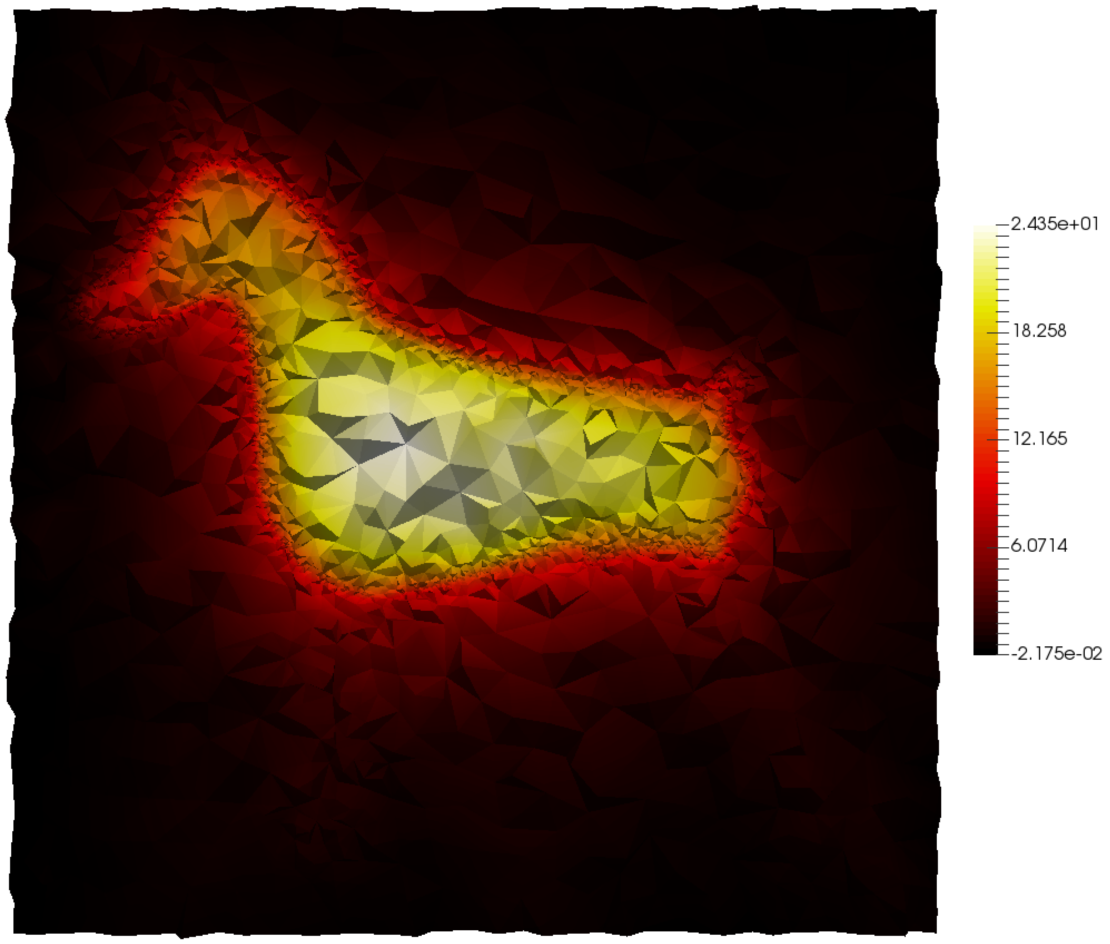


Figure 5.35: Scalar flux at the mid-plane of the Jethro-shaped source.

6. CONCLUSION

The motivations for performing the research presented in this document were presented in Chapter 1. These motivations were primarily the need for deterministic transport methods capable of handling arbitrary polyhedral spatial meshes, and the ability for implementations of such methods to take advantage of the heterogeneous nature of the next generation of super-computers. Chapter 2 presented a brief overview of the most common deterministic transport methods, and in particular highlighted the slice balance approach (SBA). The SBA was singled out for further examination due to its capability to handle extraordinarily complex meshes in a way that few other methods could. Existing implementations of the SBA however, utilized simple extensions of planar one dimensional spatial discretization schemes with relatively low accuracy. While more accurate schemes have been postulated, no evidence of their being implemented could be found as of the date of this writing.

This research began as an attempt to implement the linear discontinuous finite element (LDFE) spatial discretization scheme into the SBA framework. Along the way, two things became apparent; higher accuracy schemes would require more information on a per-slice basis than could reasonably be stored in computer memory, and a minor modification to the SBA framework itself could potentially increase accuracy in the presence of shadow-type discontinuities with relatively little added computational cost. Chapter 3 presented the extended slice balance approach (ESBA), and within the this new framework, the LDFE scheme. A face-based algorithm to perform the local sweep was presented and a slight re-definition of the slice provided the capability to divide the global mesh into independent regions, and this division resulted in the development of two new parallelization strategies.

Chapter 4 described these new parallelization strategies in detail, providing comprehensive algorithms and noting the postulated advantages and disadvantages of each. The first parallelization strategy hoped to increase scalability via added concurrency, while still keeping with the traditional method of mesh decomposition where each node of a super-computer is assigned a subset of the mesh for which it is responsible for computing and storing the angular flux solution. The second parallelization strategy strayed further from convention in that it severed the link between the node computing the solution and the node storing the solution. Chapter 4 also highlighted the dilemma imposed by the LDFE scheme, namely the large amount of information required on a per-slice basis. Storing this information in memory for arbitrary mesh and angular quadrature sizes was seemingly no longer an option as it had been for simpler differencing schemes employed by previous implementations of the SBA, such as the diamond-difference (DD) scheme.

Chapter 5 presented results generated using the Slice-T code which was written during the course of this research. The first part of Chapter 5 analyzed the accuracy of the ESBA, SBA, and the traditional cell balance approach (CBA) using the LDFE scheme and the DD like scheme presented in Grove's original implementation of the SBA.[9]. After analyzing the approaches and schemes for accuracy, the scalability of the two parallelization options made possible by the ESBA framework was examined. The acceleration of the slice and sub-slice formation using various graphics processing units (GPUs) was then reported, and finally the Slice-T code was used to solve problems of interest beyond simple manufactured solutions on uniform Cartesian hexahedral meshes. This chapter will discuss the results presented in Chapter 5, as well as highlight potential areas of future work.

6.1 Discussion

With regards to the accuracy of the ESBA compared to the SBA and CBA, the first section of Chapter 5 produced several compelling results. When the LDFE spatial discretization was used with the CBA, SBA, and ESBA, the same order of convergence was observed for a test problem with a smooth solution. With the DD scheme however, the SBA and ESBA exhibited first-order convergence, while the DD scheme with the CBA is known to be second order convergent. The absolute error however, was smaller for the ESBA than the SBA, and was smaller for the SBA than the CBA. This was the expected result for the LDFE scheme since its convergence rate for smooth solutions is independent of the shape of the volume to which it is applied, whether a slice or a cell. The fact that the convergence absolute error decreased from the CBA to the SBA is primarily a result of the increased resolution with which the solution is obtained, even though these sub-cell based solutions are combined to give the final solution on the same resolution mesh. The increase in accuracy from the SBA to the ESBA is entirely due to the resolution at which facial fluxes are computed and communicated to downstream cells and slices. In any case, the anticipated result was obtained, namely an increase in accuracy without improvement upon the rate at which the numerical solution approaches the true solution for the LDFE scheme, while the first-oder convergence of the DD scheme was determined to be a result of the shape of the slice and the location of its centroid relative to the centroids of its inlet and outlet faces.

When considering discontinuous solutions such as the propagation of a single ray or otherwise discontinuous boundary condition, the differences between the combinations of balance approach and discretization scheme were much more prominent. In this case, the ESBA outperformed the SBA and CBA in both rate of convergence and convergence coefficient when measured with a strict point-wise L^2 norm. When measuring the error in cell-averaged solution values using the absolute difference between numerical and analytical values, the SBA and CBA seemed to not converge at all, and if they were converging, it was very slowly, at least for the mesh resolutions that were practically achievable in this work. The ESBA schemes on the other hand exhibited the same convergence rates as in the case of the continuous solution. This result leads to the following conclusion; if the problem at hand is likely to exhibit shadow-type discontinuities such as particle transport through a series of ducts or various other shielding applications, and the computational resources are available, the ESBA with the LDFE scheme is likely to produce the most accurate result.

In the second section of Chapter 5, weak scaling results for the two parallelization strategies given in Chapter 4 were presented. Focusing on the first parallelization option, it seems that the added concurrency on the front end of the sweep is not enough to overcome the increase in the amount of information being communicated between adjacent nodes. This is especially true for quadrature sets containing many angles, in which case the upper limit on the parallel efficiency imposed by the idle time on the front and back ends of the sweep is already quite high. The end result in this parallelization option is that the amount of work increases over the CBA and SBA, but not by as high of a factor as the amount of data communicated, thus increasing the communication to work ratio and lowering the parallel efficiency.

The second parallelization option shows more promise. This option shows greater scalability as the number of angles per angle-set is increased. This makes sense due to the fact that the limit of placing all angles in a single angle-set results in each node performing a sweep for each angle, and the solution in each angle is independent of the solution in any other angle. This parallelization in angle only is typically not possible since in traditional methods, each node only has access to a subset of the mesh and angular flux solution, and hence cannot compute the solution for a given angle over the entire domain. Further, if there are more nodes than angles per angle-set, the amount of work is increased by increasing the overall slice count. Still, better scaling results can be obtained using this option than the first option and the volumetric decomposition method for the SBA and CBA.

When considering the next generation of super-computers where each node has access to one or more GPUs, Chapter 5 showed that the slice and sub-slice formation process can be greatly accelerated by these devices. This type of work-intensive, embarrassingly parallel application showed speedups as high as 400 for the function which identifies all the sub-slices in the mesh. Speedups as high as 130 were achieved for the sub-slice integration routine. The highest speedups were obtained using the NVIDIA V100 GPU, which is precisely the GPU used by the Sierra and Summit super-computers. Even for the NVIDIA K40 GPU used by the Titan super-computer, speedups of 223 and 38 were obtained for the sub-slice identification and integration routines respectively, and these speedups are much larger than could be obtained by multi-threading on the 16 cores of the AMD Opteron which is the CPU on each node of Titan.

Finally, Chapter 5 showed how Slice-T using the ESBA and the LDFE scheme could be used to solve more complicated problems. The first problem was a small localized source in a large volume of homogeneous dry air. This problem was used to demonstrate the method's ability to produce air transport libraries for problems exhibiting cylindrical symmetry such as air transport with a ground interface, although the interface was not included in the test problem. The other problem considered was used as a test of the method to handle a more complicated geometry. For the Jethro mesh in particular, there are plans to use this geometry as a criticality benchmark problem for codes with the capability handle such complicated geometries. One could consider the question, what density of plutonium would be required to make the Jethro mesh critical?

6.2 Future Work

In this section, we will discuss two potential areas of future work, and only briefly mention a third. The first such topic is somewhat disconnected from the SBA and ESBA, but is very much relevant to transport solutions on the next generation of super-computers. This topic is the GPU acceleration of CBA local transport sweeps using extruded prismatic meshes. During the course of this research, a similar algorithm to the one presented here was implemented for uniform Cartesian meshes and obtained speedups of roughly 30 over a single core of an Intel i7 processor using an NVIDIA GeForce GTX 870M. These results were considered promising enough that we present the more general algorithm here in hopes that it will be of interest to another researcher in the future.

The second topic of interest is the elimination of negative fluxes within the SBA and ESBA framework. Negative fluxes are the result of insufficient resolution in the spatial mesh. Since the SBA and ESBA are already resolving the mesh below the cell size and calculating a great deal of geometric information for each slice, it is thought that further refinement in the streaming direction or more creative methods requiring this geometric information could be implemented relatively easily.

A third topic that deserves further exploration is the development of a second-order DD-like spatial discretization in the SBA and ESBA frameworks. The first-order convergence of the DD scheme in the SBA and ESBA was examined in detail in Chapter 5, and it is postulated by the author that such a second-order linear method may exist. Such a method could be quite useful if for no other reason than the impressive performance of the DD scheme in the ESBA framework when applied to discontinuous solutions. If such a method could be made to converge at the second-order rate of the DD-CBA combination, it could be preferable to the standard CBA-LD treatment found in many transport codes in use today.

6.2.1 GPU Transport Sweeps for Extruded Prismatic Meshes

Why did we focus so much attention on accelerating the slice and sub-slice formation process and not the actual local sweep in Chapter 5? We did this because the preparation steps more closely resembled the SIMD nature of pixel processing, and hence were more likely to obtain the dramatic speedups that general purpose GPU (GPGPU) programming enthusiasts sometimes like to boast about. These speedups can be on the order of 10^2 or 10^3 for the right application and the right GPU. However, performing actual transport sweeps on GPUs is an active area of research[35][36], and in this section we aim to add to this discussion with a GPU implementation of the traditional CBA local transport sweep for extruded prismatic meshes. While such an application may not be likely to achieve speedups in the hundreds, a speedup of roughly 50 over a single core operating at 3 GHz would still be a big improvement, even assuming perfect linear speedup via shared memory threading on the cores of a typical 16 core node.

When it comes to the local transport sweep using the CBA, there are several possibilities for how to assign independent thread blocks on the GPU. For instance, the sweep in each energy group in the group-set is completely independent, and thus each energy group could be swept in its own thread block on its own streaming-multiprocessor (SM). This of course would require at least 15 energy groups per group-set for the NVIDIA K40 GPU, and the current trend is to make the SMs smaller and to include more of them as seen in Chapter 5. Since this seems like too stringent a requirement, we could also look to the angles, since the sweep in each angle is independent of any other. Each thread block could then be responsible for performing the sweep of a single energy group and a particular set of angles. With this in mind, we can design a kernel function to perform a sweep algorithm that could be contained to a single thread block and SM for a single energy group and angle-set.

The inspiration for the algorithm presented here is the pipe-lined Koch-Baker-Alcouffe (KBA) scheduling algorithm for orthogonal hexahedral meshes.[37] This scheduling algorithm was designed for the global sweep of a distributed mesh in which each node of the super-computer owns a columnar subset of a purely hexahedral mesh. This mesh decomposition is shown in Figure 6.1. If we further divide the mesh in the vertical dimension to obtain tasks, and index these vertical bins by h, the algorithm can be depicted as in Figure 6.2. This figure shows the first 12 stages of the sweep where each box represents a column of the mesh, divided into four vertical bins, and owned by a single node. Different angles are represented by different colors.

The algorithm is said to be pipe-lined because once the sweep plane reaches the top of the domain and the lower left node has no more work to do for the given angle, it starts on the next angle in the angle-set, which restricts all angles in the angle-set to be in the same octant of the unit sphere. Communication to the neighboring nodes in the upward and rightward direction as depicted in Figure 6.2 is required between each stage; however, communication in the vertical dimension is unnecessary because the neighboring vertical bin to each task is owned by the same node.

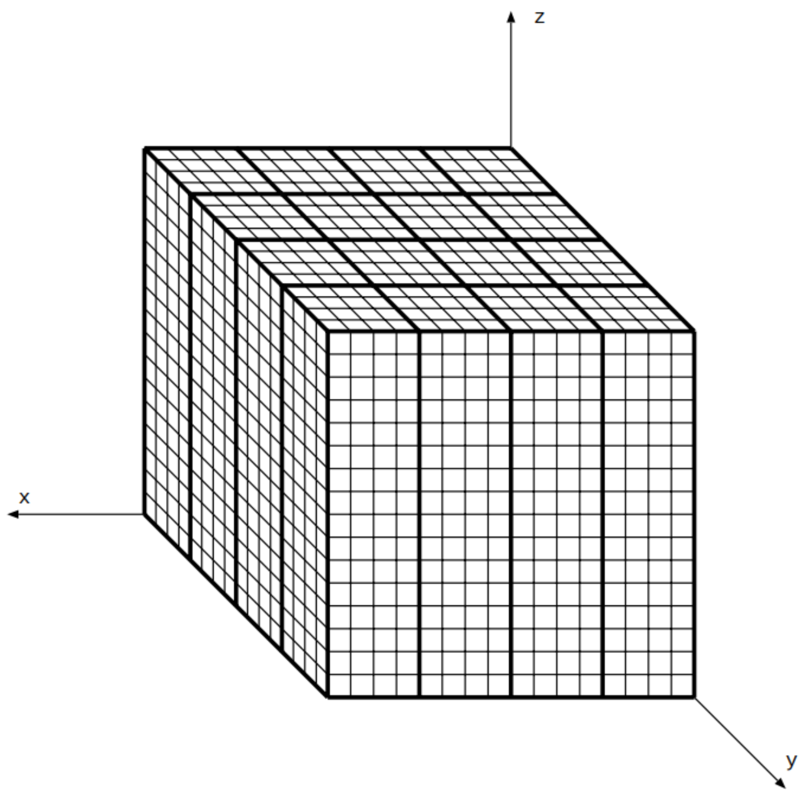


Figure 6.1: Mesh decomposition for the pipe-lined KBA scheduling algorithm. Thick lines indicate inter-processor domain boundaries while thin lines represent cell boundaries.

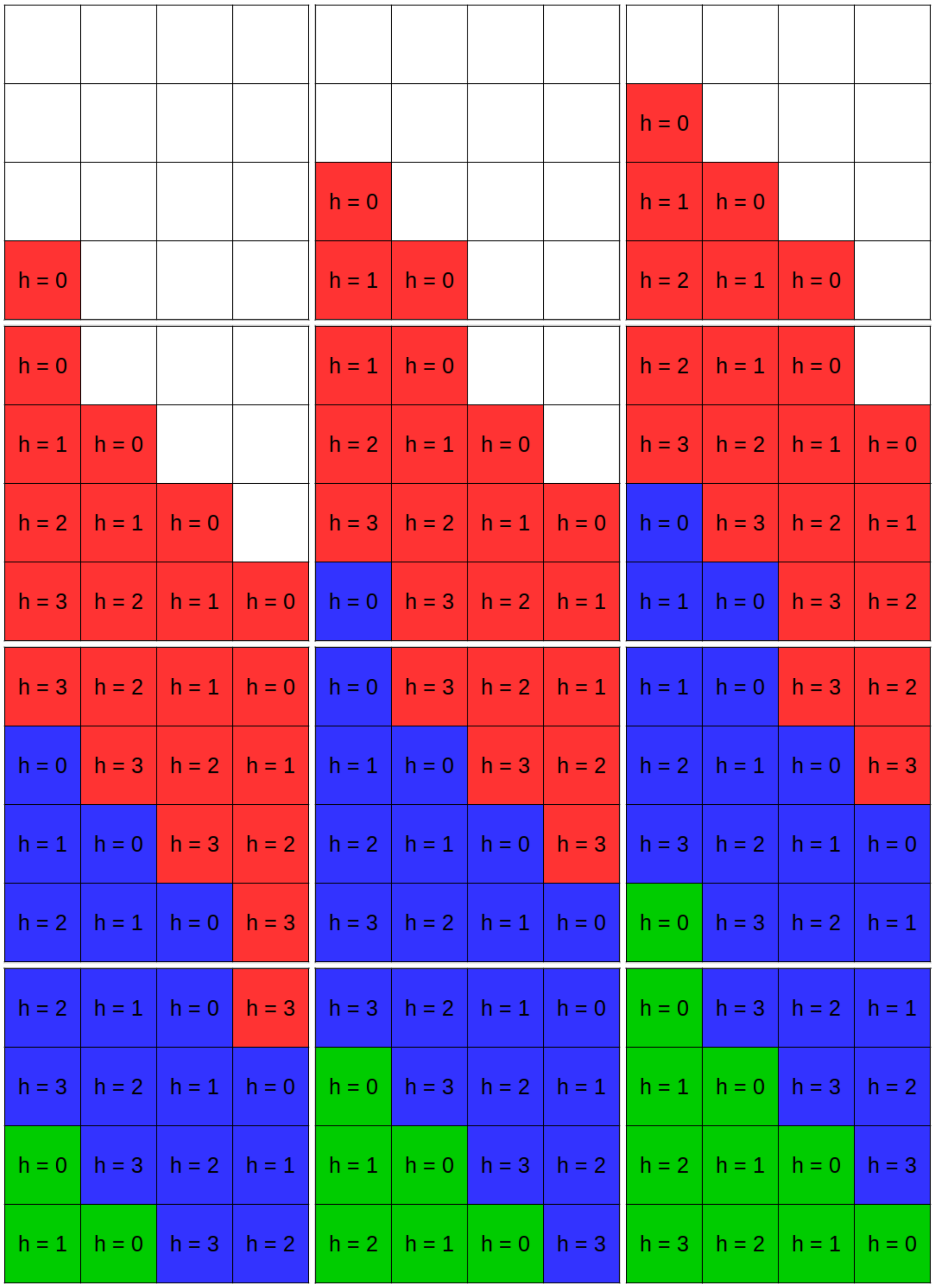


Figure 6.2: First 12 stages of the pipe-lined KBA scheduling algorithm for a sweep with 16 nodes.

The KBA scheduling algorithm for the global sweep of a distributed mesh has influenced the algorithm presented here for the local sweep on a single node with an extruded prismatic mesh. The idea is for each thread of the kernel function on the GPU to be responsible for computing the solution in a column of cells. The sweep then proceeds exactly as in Figure 6.2, with the obvious exception that the cells are no longer required to be quadrilaterals as depicted in the figure. This further restricts which angles can be included in the angle-set because all angles in the angle-set must have the same sweep dependency graph. This restriction can be satisfied for all angles in the same octant sharing the same azimuthal component.

Assigning each thread to a column of the local mesh also places a restriction on the size of the local mesh, because the GPU does not allow for an infinite number of threads per thread block. On the K40 GPU, the limit is 1,024 threads per thread block, and hence if each thread were responsible for a single column of cells, the local mesh could only contain 1,024 columns. This limitation could be removed by allowing each thread to be responsible for a group of columns.

The algorithm for the local sweep is given in Algorithm 6.1. As in most GPU kernel functions, the first step is to retrieve the thread block index b, along with the thread index within the thread block t. These are supplied via built-in functions in the CUDA C language, which is the language kernel functions are written in. The column index c can then be accessed via the thread index, since each thread will own a single column, and the energy group index g can be accessed by the thread block index, since each thread block will be performing a sweep for a single energy group. Finally the number of stages N_{stages}, can be accessed by the thread block index corresponding to the sweep of a given energy group and angle-set. Obtaining these values will require specific data structures to be passed to the GPU from the CPU.

With the thread block index, thread index, column index, energy group index, and stage count obtained, each thread then enters the stage loop. The first step of each stage is for each thread to obtain the vertical bin index h, and the angle index m, which it will be working on in this stage. These are obtained by data structures that will be supplied to the GPU by the CPU, and the values will be retrieved by providing the stage index s, the thread block index (since each thread block may have a different sweep dependency graph), and the column index. If the there is no work for the thread to do during this stage, these data structures will return an index of -1.

With the vertical bin index and angle index obtained, the next step is to build the coefficient matrix $\mathbf{A}$, and right hand side vector $\mathbf{b}$. This step should only be done by threads that have work to do during the current stage, and hence the code to perform this action is contained within an `if` statement to exclude threads that should be idle. While the $\mathbf{A}$ matrix can be built relatively easily, building the right hand side vector requires looping over all the incoming faces and adding to the vector the incoming flux moments on each incoming face.

This transfer of information from one cell to another via the face that they share in common, should ideally be through the shared memory of the SM. Unfortunately, this memory space is quite small (by default 49 kB per SM on the Kepler generation of GPUs), and hence we should make every possible effort to conserve it. Thus if there is a choice between storing a set of values for every face in the mesh, versus storing a set of values for every column of faces in the mesh, which appear as edges when viewed from above as in Figure 6.2, the choice is quite clear. Thus, we also require a data structure that can return an edge index e given a face index f, and access the angular flux coefficients from shared memory using the edge index. Once the coefficients are obtained, the angular flux moments on the incoming face can be

Algorithm 6.1: Local sweep kernel function for extruded prismatic meshes.

```
1:  $b$ = GetThreadBlockIndex()
2:  $t$ = GetThreadIndex()
3:  $c$ = GetColumnIndex($t$)
4:  $g$ = GetGroupIndex($b$)
5:  $N_{\text{stages}}$ = GetStageCount($b$)
6:  for $s = 0$ to $N_{\text{stages}} - 1$ do
7:      $h$ = GetVerticalBinIndex($s$, $b$, $c$)
8:      $m$ = GetAngleIndex($s$, $b$, $c$)
9:      if $(h \neq -1)$ and $(m \neq -1)$ then
10:         Build $\mathbf{A}$ matrix
11:         Build base of $\mathbf{b}$ vector with volumetric source moments
12:         for $i = 0$ to $N_{\text{inlet faces}} - 1$ do
13:             Retrieve face index $f$
14:             $e$ = GetEdgeIndex($f$)
15:             Retrieve flux coefficients from shared memory location $e$
16:             Add to $\mathbf{b}$ vector the incoming flux moments
17:         end for
18:     end if
19:     barrier
20:     if $(h \neq -1)$ and $(m \neq -1)$ then
21:         Solve $\mathbf{Ax} = \mathbf{b}$; a $4 \times 4$ system for the flux coefficients in the cell
22:         atomic: add volumetric angular flux moments to
23:                 global memory scalar flux moments
24:         for $i = 0$ to $N_{\text{outlet faces}} - 1$ do
25:             Retrieve face index $f$
26:             $e$ = GetEdgeIndex($f$)
27:             Store flux coefficients in shared memory location $e$
28:         end for
29:     end if
30:     barrier
31: end for
```

obtained and added to the right hand side vector $\mathbf{b}$.

Before going on to solve the system $\mathbf{Ax} = \mathbf{b}$, we should ensure that all values from shared memory have been accessed before they are overwritten after solving the system. Thus, a barrier is encountered before the next `if` statement is used to ensure threads that should be idle during this stage do not attempt to solve their systems. Within this `if` statement, each thread solves its 4×4 system, and atomically adds the volumetric angular flux moments to the corresponding volumetric scalar flux moments via the quadrature integration rule. The final step within this `if` statement is to store the angular flux coefficients on the edge indices corresponding to each outlet face of the cell. This is again performed by looping over the outlet faces, obtaining the face index f, using the supplied data structure to convert the face index to an edge index e, and then storing the flux coefficients in shared memory using the edge index. Finally, before moving on to the next stage, we again enforce a barrier to ensure that all values have been written to shared memory before any attempt to access them in the next stage are made.

6.2.2 Negative Flux Treatment

It has long been known that linear methods such as the LDFE and DD spatial discretization schemes have the propensity to produce negative fluxes for cell sizes large compared to the mean free path of the particle. Thus, if one wants to eliminate the possibility of negative fluxes, they can either refine the mesh or tweak the numerical method. The first option often leads to adaptive mesh refinement (AMR), while the second option leads to unique methods that avoid negative fluxes mathematically within the system of equations in each cell. Both avenues have been the subject of much research over the years, and it occurs to the author that the SBA provides a unique opportunity to pursue either of these strategies.

If we consider the option of mesh refinement, we should first think about why such negative fluxes occur in one-dimension. We should consider this because the one-dimensional nature of a slice that led Grove to postulate that one dimensional planar differencing schemes may be extensible to three dimensions through the SBA also leads the current author to believe that simple one-dimensional AMR techniques may be extensible as well. To illustrate why these schemes produce negative fluxes, consider Figures 6.3 through 6.6 which show the LDFE and DD approximations, alongside the exact solution, to a one-dimensional cell with an incoming unit flux on the left face and no internal source for cell sizes equal to one, two, three, and four mean free paths respectively. What these figures show is how each of these schemes attempts to approximate a decaying exponential with a simple linear function. When the range over which this approximation is made is small compared to the inverse of the exponential coefficient (i.e. $1/\sigma_t$ which is the mean free path), this approximation is made with relatively decent accuracy. As this range is increased however, the approximations become worse, and eventually fall below zero at the right boundary.

The LDFE scheme is less prone to produce negative fluxes because it does not require the flux at the left boundary to match the incoming boundary condition. Even with this benefit, it is still a linear method, and in order to preserve the zeroth and first moments of the transport equation with respect to x, it will produce negative fluxes eventually as shown in Figure 6.6. So how would one remedy this in one dimension? The simple answer is to refine the mesh until all cells are adequately small to avoid negative fluxes. How would one remedy this in three dimensions? The answer to this question is not so clear because not only would the cells need to be refined in every discrete direction, there are also the transverse slopes in the linear approximation that make the answer a little more elusive, not to mention that the refined mesh may be unnecessary for any other physics considered in the simulation.

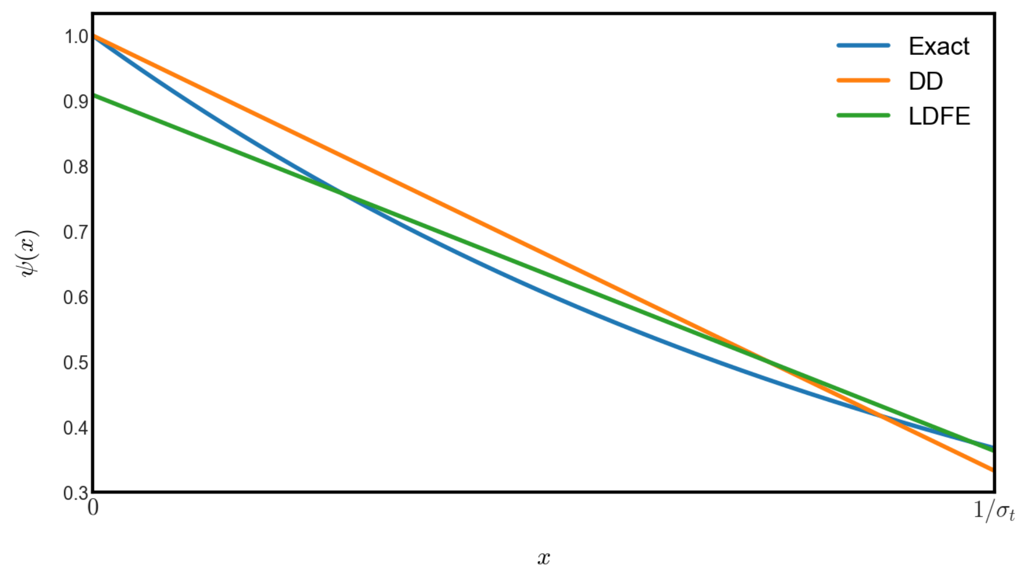


Figure 6.3: LDFE and DD approximations alongside the exact solution for a incident unit flux on the left face and no internal source for a cell of width equal to one mean free path.

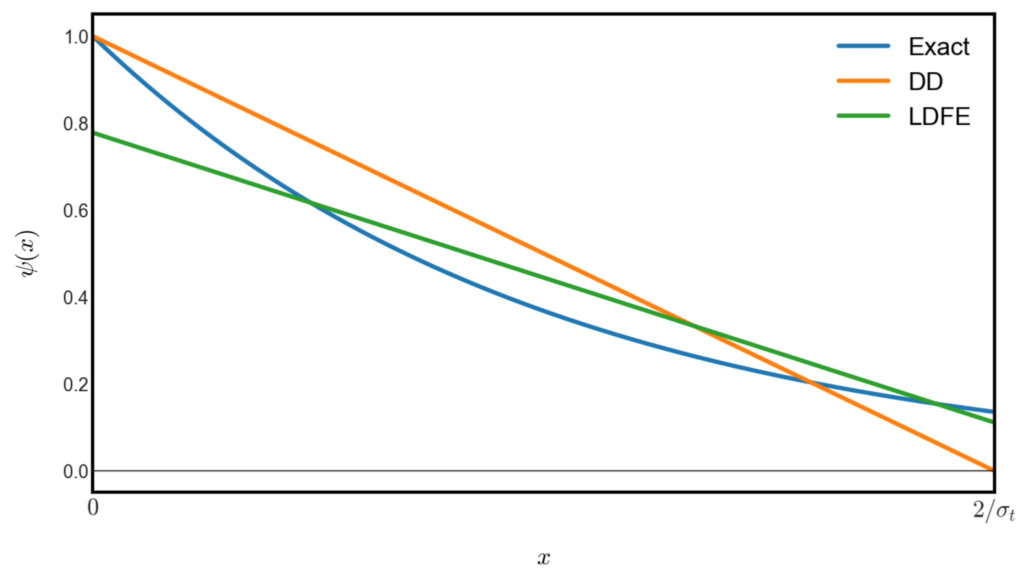


Figure 6.4: LDFE and DD approximations alongside the exact solution for a incident unit flux on the left face and no internal source for a cell of width equal to two mean free paths.

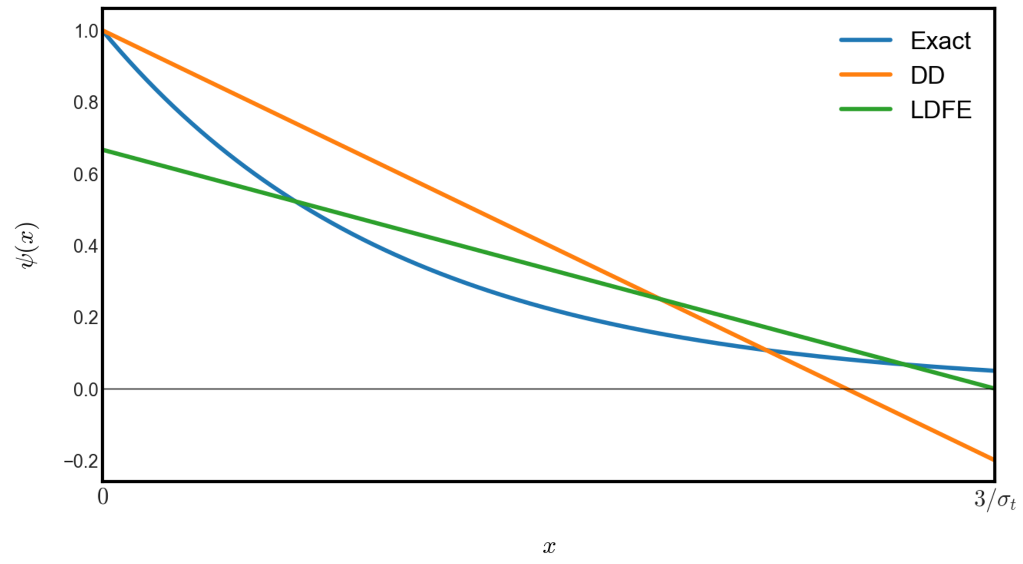


Figure 6.5: LDFE and DD approximations alongside the exact solution for a incident unit flux on the left face and no internal source for a cell of width equal to three mean free paths.

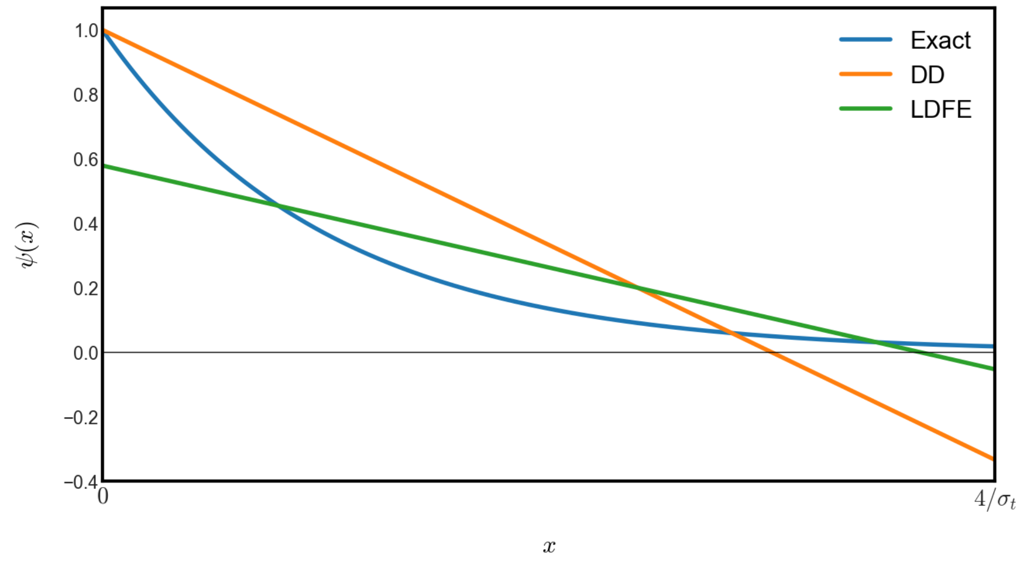


Figure 6.6: LDFE and DD approximations alongside the exact solution for a incident unit flux on the left face and no internal source for a cell of width equal to four mean free paths.

The SBA and ESBA provide a rather simple solution to this problem given the one dimensional nature of a slice. One could simply further divide the slice via planes perpendicular to the discrete ordinate into segments short enough to avoid negative outgoing fluxes. This is shown in Figures 6.7 and 6.8 which depict a typical slice, and the same slice partitioned into five segments respectively. The streaming plus collision operator could then be inverted on each partition, and then the partition flux coefficients could be combined to give the slice flux coefficients in the same way slice flux coefficients were combined to give cell flux coefficients in Chapter 3. Further, as in the slice and sub-slice formation process, this mesh refinement would be done on the fly, and would not require any actual changes be made to the global mesh which may be used by other physics which do not require such fine resolution. It is of course still possible that negative fluxes could occur in extreme cases due to the transverse slopes in the LDFE linear approximation, but it is thought that this method would be highly effective at reducing the prevalence of negative fluxes. While this technique could eliminate negative exiting fluxes, we recognize that it would not by itself prevent the cell-wise solution, which is formed from slice-wise spatial moments, from having negative values. This in turn could cause a negative collisional source, resulting in negative solutions regardless of within-slice refinement. Nevertheless, the adaptive refinement scheme outlined here might significantly reduce the frequency and severity of negative fluxes in many transport applications of practical interest.

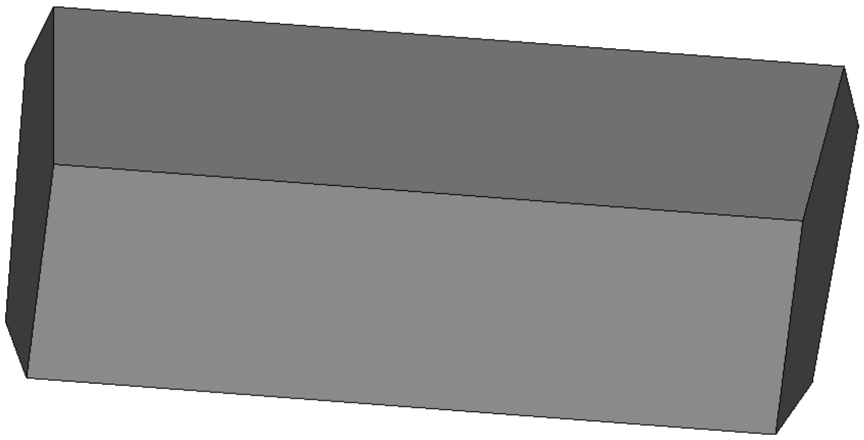


Figure 6.7: Depiction of a typical slice.

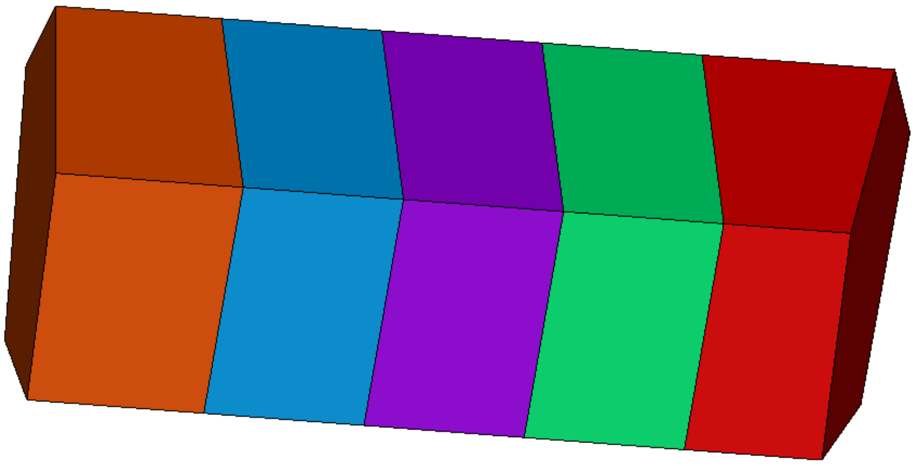


Figure 6.8: Depiction of the partitioning of the slice in Figure 6.7 into five segments along the discrete ordinate.

If we consider the second option of mathematically altering the method to avoid negative fluxes altogether, many techniques exist which either enforce a strict maximum principle or introduce an artificial viscosity. A more straightforward option might be an extension of Maginot's work[38] in which the linearity of the solution is set aside in favor piece-wise linearity with a portion of the cell flux, or slice flux in this case, set to zero, while still preserving the zeroth and first moments of the transport equation. Whichever method is chosen, it is likely to require many geometric quantities for each slice, which are already being computed.

REFERENCES

[1] *Handbook of Nuclear Engineering: Vol. 1: Nuclear Engineering Fundamentals.* Springer Verlag, Karlsruhe, Germany, 2011.

[2] E. E. Lewis and W. F. Miller. *Computational Methods of Neutron Transport.* American Nuclear Society, La Grange Park, IL, 1993.

[3] E. Fridman and J. Leppänen. On the Use of the Serpent Monte Carlo Code for Few-Group Cross Section Generation. *Annals of Nuclear Energy*, 38(6):1399 – 1405, 2011.

[4] A. Haghighat and J. C. Wagner. Monte Carlo Variance Reduction with Deterministic Importance Functions. *Progress in Nuclear Energy*, 42(1):25 – 53, 2003.

[5] M. J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Trans. Comput.*, 21(9):948–960, September 1972.

[6] Technology Quarterly: After Moore's Law. *The Economist*, 2016.

[7] I. Buck, J. Nichols, and R. Neely. GPU Acceleration: What's Next? In *SC14*, New Orleans, LA, 2014.

[8] Top 500, June 2018. `https://www.top500.org/lists/2018/06/`. Accessed: 2018-10-26.

[9] R. E. Grove. *A Characteristic-based Multiple Balance Approach for Solving the S_N Equations on Arbitrary Polygonal Meshes.* PhD thesis, University of Michigan, 1996.

[10] M. Peric. Flow Simulation Using Control Volumes of Arbitrary Polyhedral Shape. *ERCOFTAC Bulletin*, (62), September 2004.

[11] M. Spiegel, T. Redel, Y. J. Zhang, T. Struffert, J. Hornegger, R. G. Grossman, A. Doerfler, and C. Karmonik. Tetrahedral vs. Polyhedral Mesh Size Evaluation on Flow Velocity and Wall Shear Stress for Cerebral Hemodynamic Simulation. *Computer Methods in Biomechanics and Biomedical Engineering*, 14(1):9–22, 2011.

[12] G. Balafas. Polyhedral Mesh Generation for CFD-Analysis of Complex Structures. Master's thesis, Technische Universitt Munchen, Germany, 2014.

[13] D. Kincaid and W. Cheney. *Numerical Analysis: Mathematics of Scientific Computing (Second edition)*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 1996.

[14] S. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods*. Texts in Applied Mathematics. Springer New York, 2007.

[15] J.R. Askew. *A Characteristics Formulation of the Neutron Transport Equation in Complicated Geometries*. AEEW-M. Atomic Energy Etablishment, 1972.

[16] J. E. Morel and E. W. Larsen. A Multiple Balance Approach for Differencing the S_N Equations. *Nuclear Science and Engineering*, 105(1):1–15, 1990.

[17] M. L. Adams and E. W. Larsen. Fast Iterative Methods for Discrete-Ordinates Particle Transport Calculations. *Progress in Nuclear Energy*, 40(1):3 – 159, 2002.

[18] W. D. Hawkins, T. Smith, M. P. Adams, L. Rauchwerger, N. M. Amato, and M. L. Adams. Efficient Massively Parallel Transport Sweeps. *Trans. Amer. Nucl. Soc*, 107:477–481, 2012.

[19] M. P. Adams, M. L. Adams, W. D. Hawkins, T. Smith, L. Rauchwerger, N. M. Amato, T. S. Bailey, and R. D. Falgout. Provably Optimal Parallel Trans-

port Sweeps on Regular Grids. In *Proceedings of the American Nuclear Society Mathematics and Computation Conference*, MC2013. ANS, 2013.

[20] S. D. Pautz. An Algorithm for Parallel S_N Sweeps on Unstructured Meshes. *Nuclear Science and Engineering*, 140(2):111–136, 2002.

[21] G. Colomer, R. Borrell, F.X. Trias, and I. Rodrguez. Parallel Algorithms for S_N Transport Sweeps on Unstructured Meshes. *Journal of Computational Physics*, 232(1):118 – 135, 2013.

[22] T. H. Ghaddar. Load Balancing Unstructured Meshes for Massively Parallel Transport Sweeps. Master's thesis, Texas A&M University, 2016.

[23] R. A. Kennedy, A. M. Watson, and R. E. Grove. Linear Discontinuous (LD) Coefficients In The Slice Balance Approach (SBA) Mathematical Framework For The Discrete Ordinates Code Jaguar. In *Proceedings of PHYSOR 2010*, LaGrange Park, IL, 2010. American Nuclear Society.

[24] F. Pellegrini and J. Roman. Scotch: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs. In *High-Performance Computing and Networking: International Conference and Exhibition HPCN EUROPE 1996 Brussels, Belgium, April 15–19, 1996 Proceedings*, pages 493–498. Springer Berlin Heidelberg, 1996.

[25] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[26] I. E. Sutherland and G. W. Hodgman. Reentrant Polygon Clipping. *Commun. ACM*, 17(1):32–42, January 1974.

[27] OpenCFD. OpenFOAM User Guide. *OpenFOAM Foundation*, 2(1), 2011.

[28] A. C. Kahler, R. E. MacFarlane, D. W. Muir, and R. M. Boicourt. The NJOY Nuclear Data Processing System, Version 2012. *Los Alamos National Laboratory*, 2012.

[29] K. A. Mathews. On the Propagation of Rays in Discrete Ordinates. *Nuclear Science and Engineering*, 132(2):155–180, 1999.

[30] NVIDIA. Kepler GK110. The Fastest, Most Efficient HPC Architecture Ever Built. Technical report, NVIDIA Corporation, 2012.

[31] NVIDIA. NVIDIA Tesla P100. The Most Advanced Data Center Accelerator Ever Built Featuring Pascal GP100, the Worlds Fastest GPU. Technical report, NVIDIA Corporation, 2016.

[32] NVIDIA. NVIDIA Tesla V100 GPU Architecture. The World's Most Advanced Data Center GPU. Technical report, NVIDIA Corporation, 2017.

[33] J. J. Burgio. Nuclear Radiation Dose Comparisons of SMAUG to Air Transport Results Based on the DLC-31 Cross Sections. Technical report, Air Force Weapons Lab Kirtland AFB New Mexico, 1977.

[34] K. R. DePriest and P. J. Griffin. NuGET Users Guide: Revision 0. *SAND Report, SAND2004-1567 (OUO/ECI), Sandia National Laboratories, Albuquerque, NM*, 2004.

[35] C. Gong, J. Liu, L. Chi, H. Huang, J. Fang, and Z. Gong. GPU Accelerated Simulations of 3D Deterministic Particle Transport Using Discrete Ordinates Method. *Journal of Computational Physics*, 230(15):6010 – 6022, 2011.

[36] D. S. Efremenko, D. G. Loyola, A. Doicu, and R. J. D. Spurr. Multi-core-CPU and GPU-accelerated Radiative Transfer Models Based on the Discrete Ordinate Method. *Computer Physics Communications*, 185(12):3079 – 3089, 2014.

[37] K. R. Koch, R. S. Baker, and R. E. Alcouffe. A Parallel Algorithm for 3D $\mathrm{S_N}$ Transport Sweeps. Technical Report LA-CP-92-406, Los Alamos National Laboratory, 1992.

[38] P. G. Maginot. A Nonlinear Positive Extension of the Linear Discontinuous Spatial Discretization of the Transport Equation. Master's thesis, Texas A&M University, 2010.

[39] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, pages 203–222. Springer, 1996.

[40] H. Si. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.*, 41(2):11:1–11:36, February 2015.

APPENDIX A

BRICK DECOMPOSITION AND LOAD BALANCING

As mentioned in Chapter 2, one drawback to the traditional transport sweep is the requirement for inter-node domain boundaries to be planar in order to avoid ray re-entry. Non-planar inter-node domain boundaries will likely introduce cycles in the sweep dependency graph, requiring further iteration inside the inner-most iterations, where parallelism is employed to solve high resolution transport problems. One might ask, why is this requirement a bad thing? The answer comes from anyone who has ever tried to mesh a complex geometry. It is simply not the case that such planar separations will always exist within the problem geometry, and hence they must be introduced by the user of the meshing software. Further, even if there are such naturally occurring planar separations in the geometry of interest, there is no guarantee that these planes will separate the mesh such that the number of cells in each subset of the mesh is roughly equal or load-balanced.

Load-balancing is extremely important for the scalability of parallel deterministic transport solutions. If one node contains a portion of the mesh that is much larger than the portions given to any other node, it is likely that the other nodes will finish sooner than the load-bearing node, leading to idle time as they wait for this node to finish. Thus, the problem is really two-fold. The first objective is to divide the mesh such that all inter-node domain boundaries are planar, while the second objective is to divide the mesh as equally as possible such that all nodes receive roughly the same number of spatial cells. To this end, Ghaddar [22] has implemented a load-balanced meshing algorithm for triangular extruded meshes using the Triangle[39] software for use in the transport code Parallel Deterministic Transport (PDT).[19]

The algorithm presented in this appendix is simply a recursive analog of a one dimensional version of Ghaddar's algorithm. The primary difference between the two implementations is how the planar separations are placed into the mesh. In Ghaddar's work, these separating planes are used to define the geometry that is re-meshed at each iteration of the load-balancing algorithm, while in this work the original mesh is simply refined by the inserted planes which cut through the original cells of the mesh. This has the drawback of possibly creating some very poorly shaped cells, but has the advantage of being independent of the mesh type and requiring no re-meshing. Having said this, we will assume that the user is able to perform this insertion, while recognizing that it is by no means a trivial task, and is likely the most difficult part of the algorithm to implement. One reason this plane insertion is not described in detail is that the algorithm to do so greatly depends upon how the mesh is stored in computer memory.

To illustrate how to choose the location of the dividing planes, consider the cumulative density function given in Figure A.1. This function represents the fraction of cells that have any portion of their volume below a given x value. This plot is an idealization, and in reality the cumulative density function in any Cartesian dimension is unlikely to be quite so smooth, but it is at least guaranteed to be monotonically increasing. If we knew this density function, choosing the location of the separating planes would be quite simple. For instance, if we wanted to divide the mesh into three regions with equal numbers of cells, we would find the x locations that correspond to $F(x) = 0.\overline{3}$ and $F(x) = 0.\overline{6}$ as shown in Figure A.2. Of course, we don't know this density function, and we must approximate it somehow. One could simply calculate $F(x)$ for many values of x, or guess the separating plane locations and iteratively adjust them until a suitable load-balancing metric is achieved. We choose the second option.

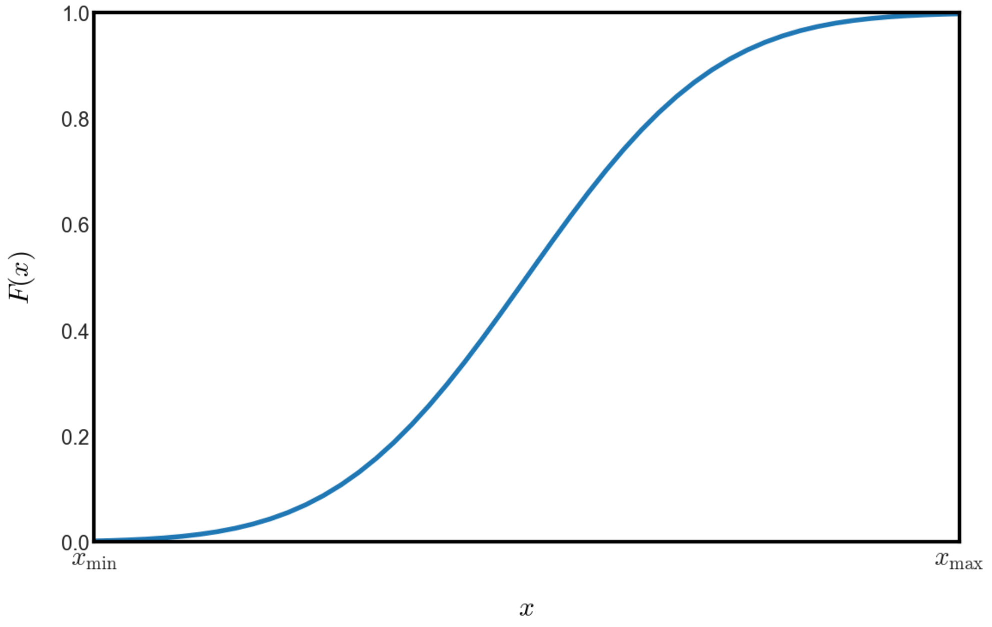


Figure A.1: Cumulative density function for the number of cells in the x dimension.

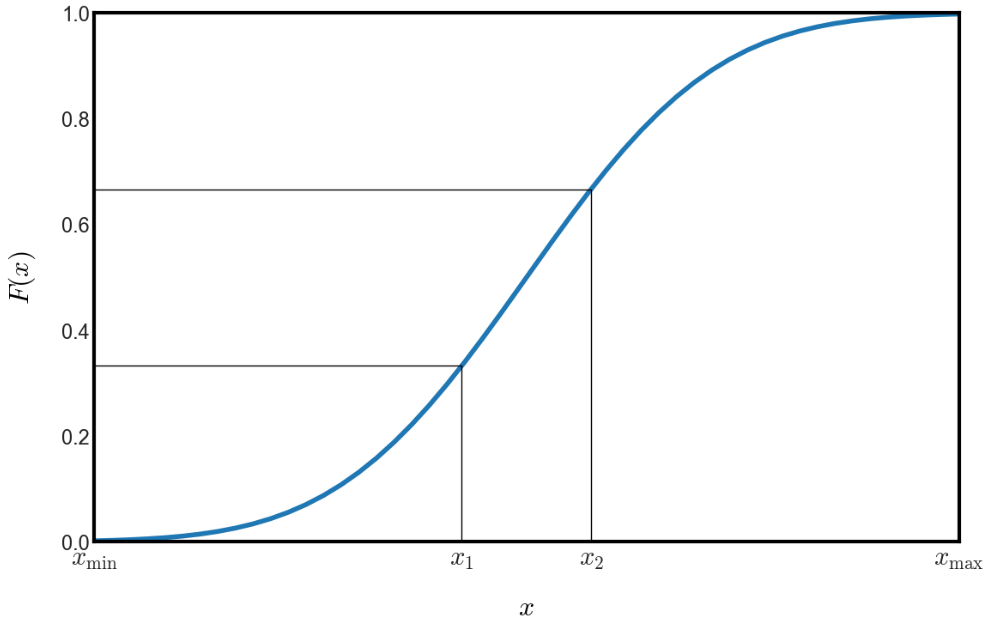


Figure A.2: Illustration of how to choose the locations of the dividing planes using the cumulative density function from Figure A.1.

To illustrate how the iterative procedure works, consider the following problem. We want to split the mesh into four bins in the x dimension. We first assume that the cells of the mesh are equally distributed so that the three internal separating planes are equally spaced between x_{min} and x_{max}, and then calculate the value of $F(x)$ at each location. We then connect these data points via linear interpolation, and use this piecewise linear function to simply read off the x values corresponding to $F(x) = 0.25$, 0.5, and 0.75. This is shown in Figure A.3. We then move the separating planes to these new locations, and iterate the process until a suitable load-balancing metric is achieved. It should be noted that the complicated process of inserting the separating planes into the mesh only needs to be started once the iterative process has converged. The second iteration is depicted in Figure A.4, and the algorithm to perform this iterative procedure is given in Algorithm A.1.

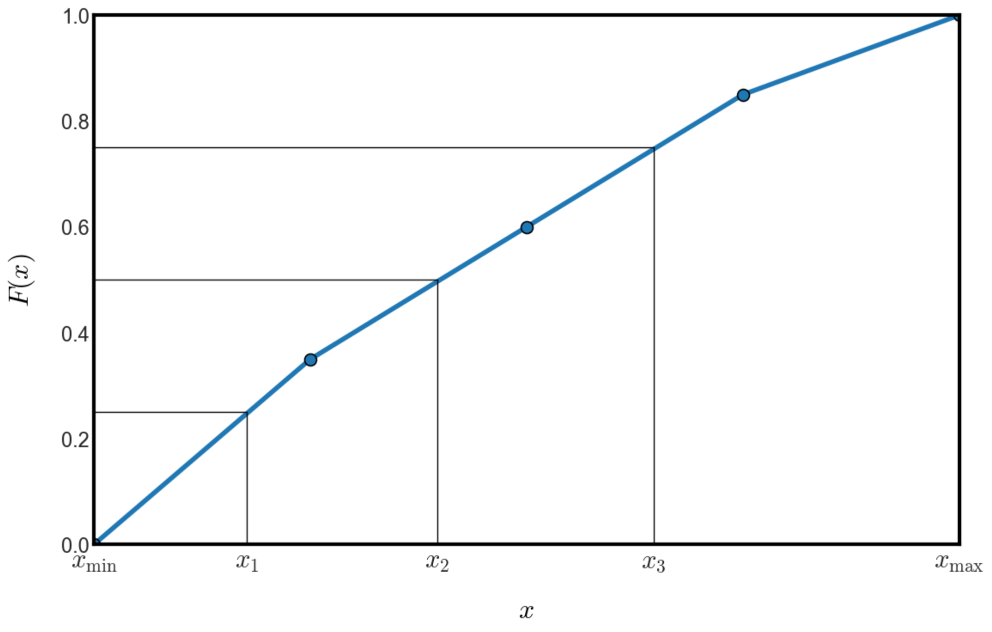


Figure A.3: Illustration of choosing the separating plane locations in the first iteration given a uniform initial guess.

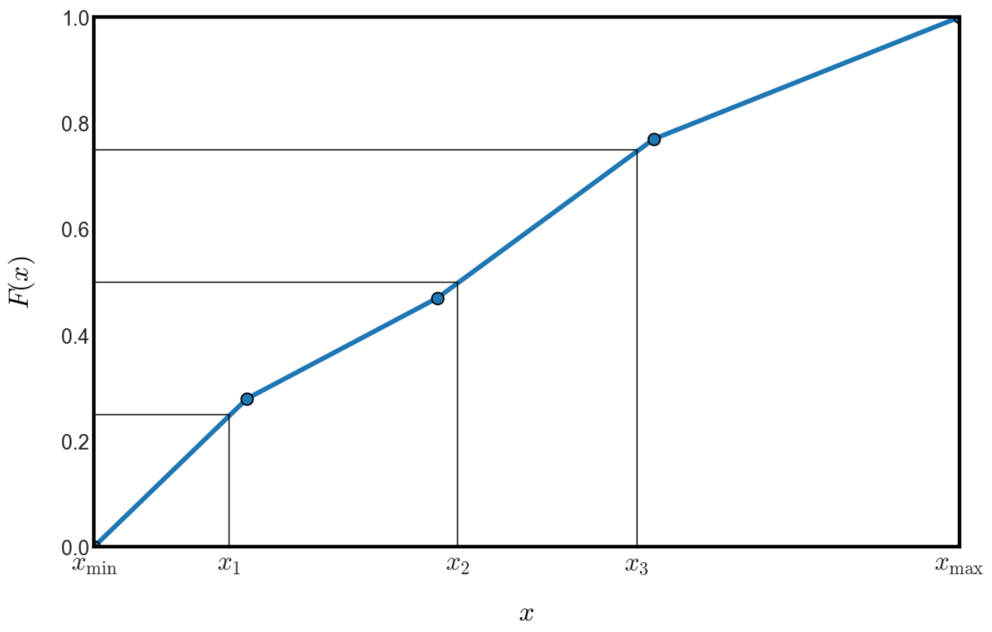


Figure A.4: Illustration of the second iteration choosing new separating plane locations.

Now that we have described how to load balance in a single dimension via the iterative procedure described above, we must confront the reality that in nearly all cases of interest, division of a three dimensional mesh in a single Cartesian dimension is not the desired goal. The desired goal is to divide the three dimensional mesh into brick-shaped sub-domains, each containing roughly equal numbers of cells. The trick to extending this algorithm to three dimensions is recursion. We first divide the mesh in the x dimension into N_x bins. We then take each of these N_x sub-domains and divide each one in the y dimension into N_y bins. At this point we will have $N_x \times N_y$ sub-domains, each of which is then divided into N_z bins in the z dimension. Also note that there is no reason to separate and load-balance in the order x, y, and then z. The order could have just as easily been reversed, and in any implementation should be left as an option for the user.

Algorithm A.1: Iterative procedure for choosing the separating plane locations in order to load balance the mesh in the x dimension.

```
1: Set initial plane locations evenly spaced between $x_{\min}$ and $x_{\max}$
2: Label these values $x_0 = x_{\min},\ x_1,\ x_2, \ldots\ x_{N_{\text{bins}}} = x_{\max}$
3: $f_{\text{ideal}} = 1/N_{\text{bins}}$
4: for $k = 0$ to $N_{\text{Max iterations}} - 1$ do
5:     for $b = 0$ to $N_{\text{bins}} - 1$ do
6:         Count the number of cells residing in bin $b$: $N_{c,b}$
7:     end for
8:     $N_{c,\text{sum}} = \sum_{b=0}^{N_{\text{bins}}-1} N_{c,b}$
9:     if Load-balancing metric is acceptable then
10:        break from the $k$ loop
11:    end if
12:    $F_0 = 0$
13:    for $b = 0$ to $N_{\text{bins}} - 1$ do
14:        $F_{b+1} = N_{c,b}/N_{c,\text{sum}} + F_b$
15:    end for
16:    for $b = 0$ to $N_{\text{bins}} - 1$ do
17:        $m_b = (F_{b+1} - F_b)/(x_{b+1} - x_b)$
18:    end for
19:    $\zeta_0 = x_0 \quad ; \quad \zeta_{N_{\text{bins}}} = x_{N_{\text{bins}}}$
20:    for $b = 1$ to $N_{\text{bins}} - 1$ do
21:        $f = b\, f_{\text{ideal}}$
22:        for $j = 1$ to $N_{\text{bins}}$ do
23:            if $f < F_j$ then
24:                $\zeta_b = x_j + (f - F_j)/m_{j-1}$
25:                break from the $j$ loop
26:            end if
27:        end for
28:    end for
29:    for $b = 0$ to $N_{\text{bins}}$ do
30:        $x_b = \zeta_b$
31:    end for
32: end for
```

The full algorithm for dividing the three dimensional mesh into brick shaped subdomains, each containing roughly equal numbers of cells is given in Algorithm A.2. The algorithm begins by defining two sets of cell groups, G_{old} and G_{new}. At the beginning of the process, G_{new} is empty and G_{old} contains the indices of all cells in the mesh. The algorithm then enters a loop over the three dimensions, and chooses which dimension to load-balance in during each iteration based on user input. The next step is to get the number of bins to split the mesh into for the given dimension, $N_{\text{bins},i}$ also based on user input. The next step is to loop over the old cell groups, which for the first dimension will be of size one. At this point, Algorithm A.1 is used to determine the location of the separating planes within this group. Once the plane locations are chosen, the algorithm builds a list of all cells that straddle these planes, and uses the planes to cut these cells, with all newly formed cells placed into the current group g. With the cells in the current group split by the separating planes, the algorithm then determines which bin each cell of the old group belongs to, and places them in the new cell groups. Finally, before moving onto the next dimension, the new groups are copied into the old groups, and the new groups are removed and resized to zero.

In order to demonstrate the efficacy of the algorithm described above, we will be using the very complicated geometry shown in Figure A.5. This figure shows a mesh of a Star Wars X-Wing inside of a bounding box, with the inside and outside of the ship meshed with different resolutions. The mesh contains 2,513,629 cells ranging in volume from 6.138×10^{-12} to 3.955, and was produced using the Tetgen tetrahedral meshing software.[40] The geometry in Figure A.5 is chosen specifically to illustrate the difficulty of the task at hand. The only obvious plane of separation which would lead to two load-balanced subsets is the plane of symmetry through the center of the ship, and even this plane does not exist in the original mesh.

Algorithm A.2: Separation of a three dimensional mesh into load-balanced brick shaped sub-domains.

```
1: $G_{\text{old},0}$ = a group of the indices of all cells in the mesh, called a cell group
2: $G_{\text{new}}$ = an empty array of cell groups
3: **for** $i = 0$ to 2 **do**
4:     $d = i^{th}$ dimension in which to load-balance $= x$, $y$, or $z$
5:     $N_{\text{bins},i}$ = the number of bins in which to divide the mesh in the $i^{th}$ dimension
6:     **for** $g = 0$ to $N_{G,\text{old}} - 1$ **do**
7:         Use Algorithm A.1 to get the plane locations $\{\xi_k\}_{k=0}^{N_{\text{bins},i}}$, in dimension $d$
8:         Build a list of cells which straddle each of the planes
9:         Use the planes to split these cells, adding each new cell index to $G_{\text{old},g}$
10:        $N = N_{G,\text{new}}$
11:        Resize $G_{\text{new}}$ to be of size $N + N_{\text{bins},i}$
12:        **for** $j = 0$ to $N_{c,\text{old},g} - 1$ **do**
13:            Get the cell index and determine which bin $b$ it belongs to
14:            Add the cell index to $G_{\text{new},N+b}$
15:        **end for**
16:    **end for**
17:    Copy $G_{\text{new}}$ to $G_{\text{old}}$
18:    Resize $G_{\text{new}}$ to size 0
19: **end for**
```

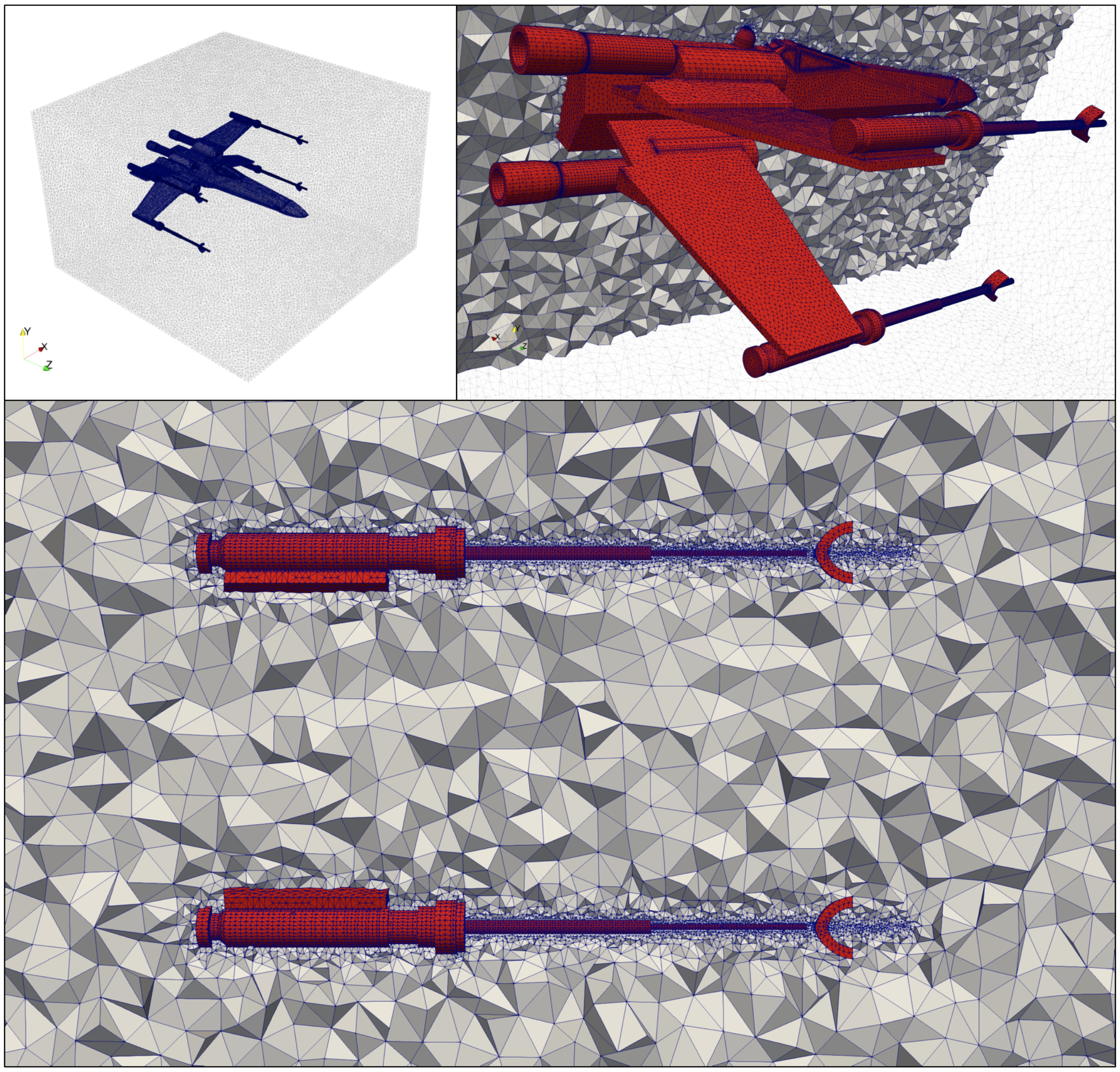


Figure A.5: Mesh of a Star Wars X-Wing used to demonstrate the load-balancing brick decomposition algorithm. The upper left shows the X-Wing inside of the bounding box with the space around the X-Wing removed. The upper right shows the surrounding mesh clipped half-way through the X-Wing to show the difference between the X-Wing mesh resolution and the surrounding space resolution. The bottom shows a clip of the surrounding mesh through the tip of the wings to show the range of mesh resolution from coarsest to finest.

Figures A.6, A.7, and A.8 show the separating planes introduced after load-balancing in the x, y, z dimensions respectively, using four bins in each dimension.

Each subset in these figures is represented by a different color. The final result is a division of the original mesh into 64 brick-shaped subsets with roughly equal numbers of cells. The load-balancing metric used in this case was to take the subset with the largest number of cells and determine how much larger this number of cells is than the average number of cells in each subset. The result for this metric in the case of the X-Wing mesh was 18.9% above the average. This is by far the highest metric encountered for any mesh up to the point of this writing, however considering the geometry at hand, this is considered quite phenomenal.

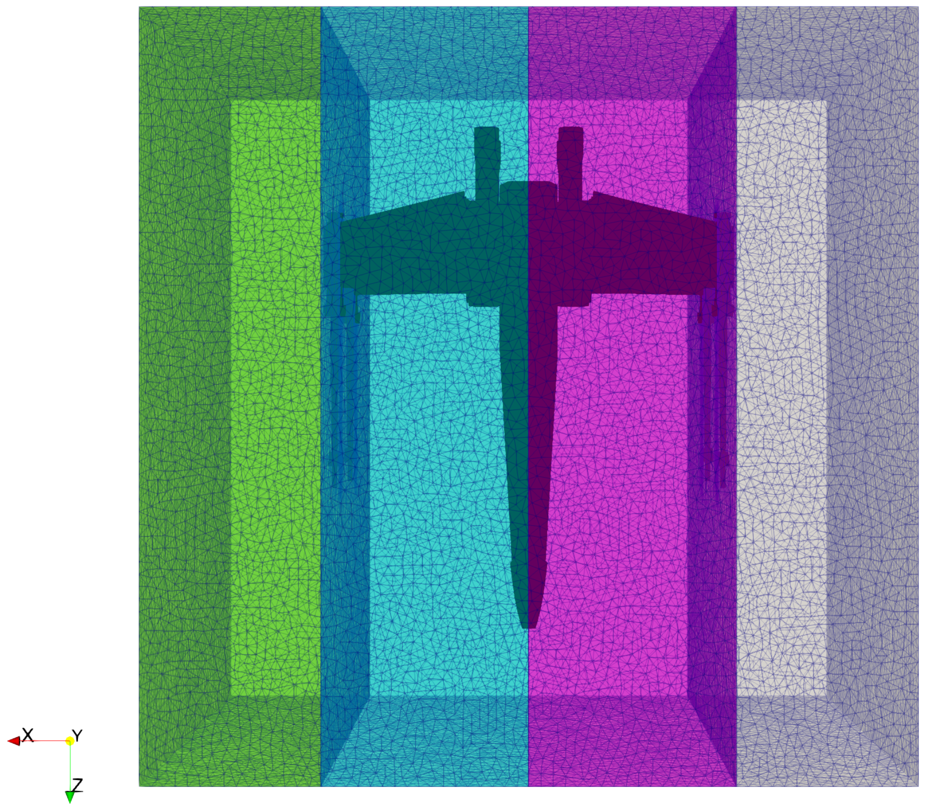


Figure A.6: Mesh after division in the x dimension. Different colors represent different subsets of the mesh assigned to nodes of a cluster or super-computer.

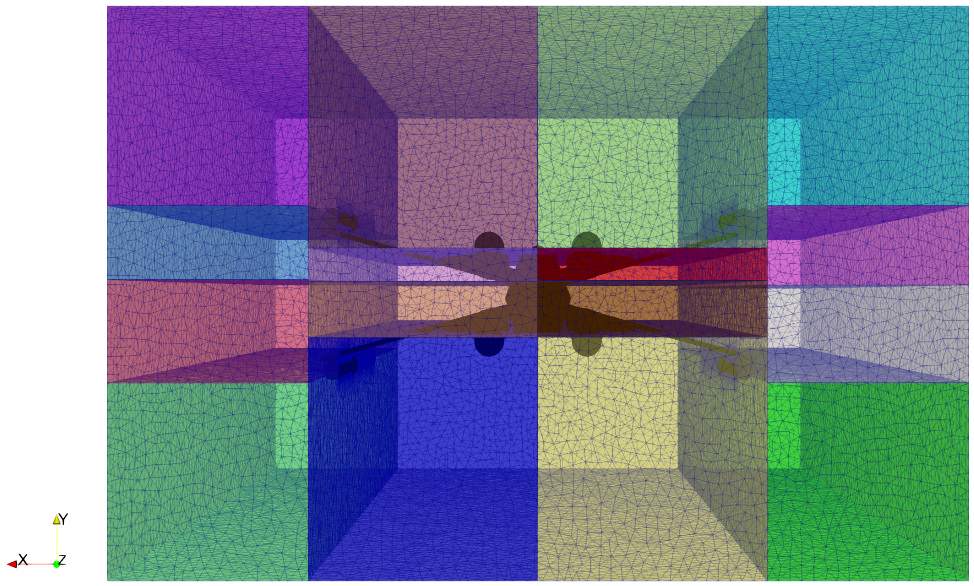


Figure A.7: Mesh after division in the y dimension. Different colors represent different subsets of the mesh assigned to nodes of a cluster or super-computer.

After separating in the x dimension, the algorithm correctly locates the plane of symmetry through the center of the ship, as expected. The other two internal separating planes appear to separate the tips of the wings, where the highest resolution is located, from the body of the ship, which makes intuitive sense as well. After separating in the y dimension, a similar theme is encountered, and the separating planes appear to be located near refined features of the ship that lead to high local cell densities. This is expected because a local high cell density should be partitioned to different subsets if the final mesh is to be load-balanced. This trend is not as obvious after separating in the z dimension from Figure A.8, but it is certainly still occurring.

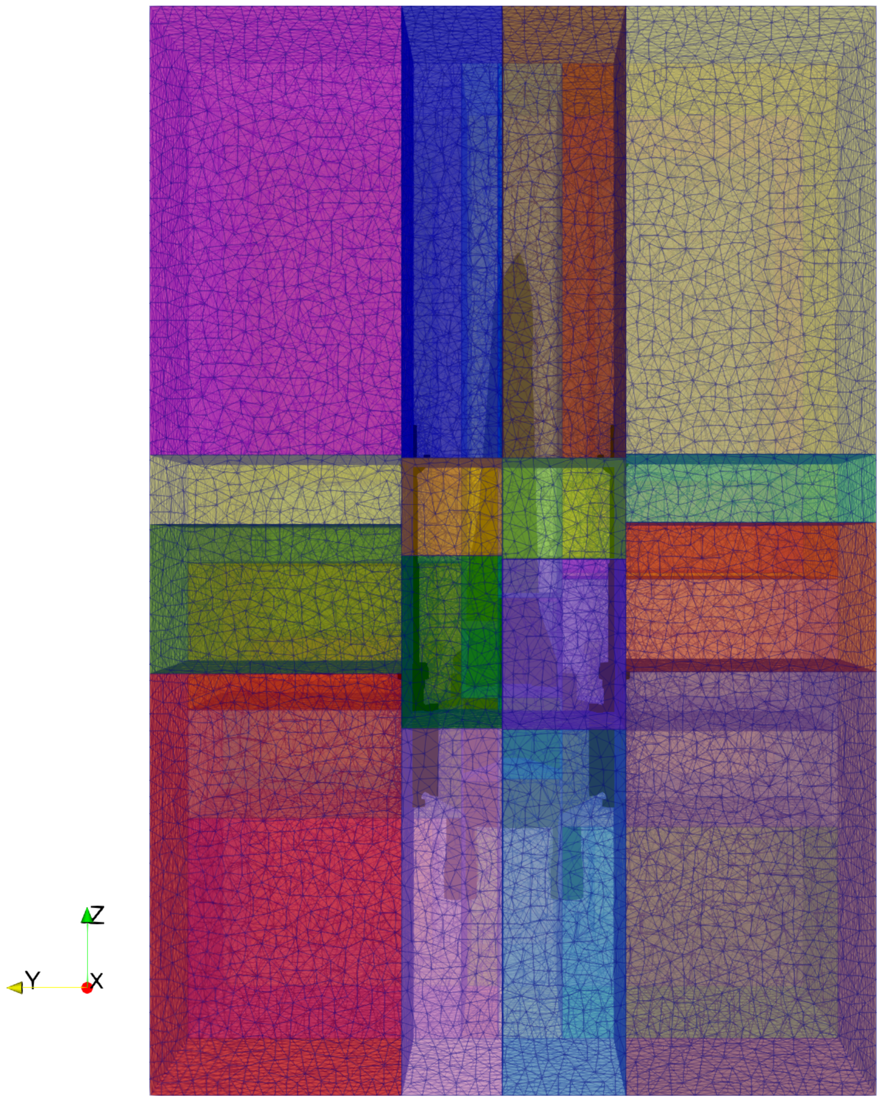


Figure A.8: Mesh after division in the z dimension. Different colors represent different subsets of the mesh assigned to nodes of a cluster or super-computer.