5-2013

Semi-Implicit Direct Kinetics Methodology for Deterministic, Time-Dependent, Three-Dimensional, and Fine-Energy Neutron Transport Solutions

James Ernest Banfield
University of Tennessee - Knoxville, jbanfie1@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by James Ernest Banfield entitled "Semi-Implicit Direct Kinetics Methodology for Deterministic, Time-Dependent, Three-Dimensional, and Fine-Energy Neutron Transport Solutions." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Nuclear Engineering.

G. Ivan Maldonado, Major Professor

We have read this dissertation and recommend its acceptance:

Kevin Clarno, Brian Wirth, Robert Grzywacz, Bobby Philip

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Semi-Implicit Direct Kinetics Methodology for Deterministic, Time-Dependent, Three-Dimensional, and Fine-Energy Neutron Transport Solutions

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

James Ernest Banfield

May 2013

ACKNOWLEDGEMENTS

Some of the work presented in this dissertation was supported by a research contract between the University of Tennessee Nuclear Engineering Department and the Reactor and Nuclear Systems Division of the Oak Ridge National Laboratory, which is managed by UT-Battelle for the United States Department of Energy (DOE). In addition, the author of this dissertation was supported by a DOE Nuclear Engineering University Programs (NEUP) three-year fellowship received during his graduate studies at the University of Tennessee.

I would like to thank my professor, G. Ivan Maldonado, and my technical mentor, Kevin Clarno, for all of their hard work and support during my dissertation. I would also like to thank the entire Advanced Multi-Physics Team for all of their contributions and support including: Kevin Clarno, Bobby Philip, Steven Hamilton, Markus Piro, Bill Cochran, Rahul Sampath, Srikanth Allu, Pallab Barai, Srdjan Simunovic, Larry Ott, Sreekanth Pannala, Phani Nukala, Gary Dilts, Bogdan Mihaila, Cetin Unal, Gokhan Yesilyurt, Jungho Lee, Mark Berrill, Mark Baird, John Turner, David Pugmire, Aaron Phillipe, and Thomas Evans. I would also like to thank my fellow students; Shane Hart for his invaluable assistance in compiling software, and Brenden Mervin for his assistance with the Exnihilo package.

ABSTRACT

Using a semi-implicit direct kinetics (SIDK) method that is developed in this dissertation, a finer neutron energy discretization and improved fidelity for transient radiation transport calculations are facilitated to reduce uncertainties and conservatisms in transient power and temperature predictions. These capabilities are implemented within a parallel computational solver framework, which is able to represent an arbitrary number of neutron energy groups, angles, and spatial discretizations, while internally coupled to an unstructured finite element multi-physics code for temperature and displacement calculations. This capability is demonstrated on a three-dimensional control rod ejection simulation run in parallel utilizing forty-four neutron energy groups.

An improved transient nuclear reactor simulation capability is developed by adapting the steady-state radiation transport code Denovo to solve the time-dependent Boltzmann transport equation for transient power distributions. The developed SIDK method is compared to fully-implicit direct kinetics, higher order time integration methods, as well as various computational benchmarks. Errors resulting from time integration, spatial discretization, angular treatment, multi-group treatment, homogenization of temperature, and power over the time step representation are explored.

For verification, the SIDK method is developed and tested externally and independently employing a few-group time-dependent neutron diffusion code which is compared to one and two-dimensional benchmarks with and without temperature feedbacks. The results of the semi-implicit direct kinetics method (SIDK) are shown to be accurate to within ~0.2% of direct kinetics and to execute roughly an order of magnitude faster, using a consistent space and time discretization. For sufficiently severe transients, the direct method is shown to produce lower errors with medium time steps than the SIDK method with fine steps, but proves to be subject to more severe oscillations at very coarse time steps than the SIDK method, in addition to producing similar errors (within 0.2 %) at medium spatial discretization with consistent time steps.

The objective of this dissertation is to provide developers of next generation high-performance computing neutron kinetics methods a guide to the benefits and costs of the dominant discretization strategies of time, space, neutron energy, and angle for the solution of the time-dependent Boltzmann transport equation.

Table of Contents

List of Figures

List of Tables

Chapter 1 INTRODUCTION

Nuclear power supplies approximately 20% of the electrical power generation in the United States, which is currently produced by 104 operating commercial Light Water Reactors (LWRs). The most fundamental components within the core of a reactor -where the heat from nuclear fission is generated- are the nuclear fuel elements, also referred to as fuel rods or fuel pins, which contain the fissile material in the form of low enriched uranium dioxide sintered pellets. Ensuring a safe and reliable operation of every single fuel pin in a reactor is the combined responsibility of fuel vendors, utilities, and regulators. Figure 1 below illustrates the large range of physical dimensions involved between a reactor vessel, the core, the fuel assemblies, and the individual pins representative of a typical Boiling Water Reactor (BWR).

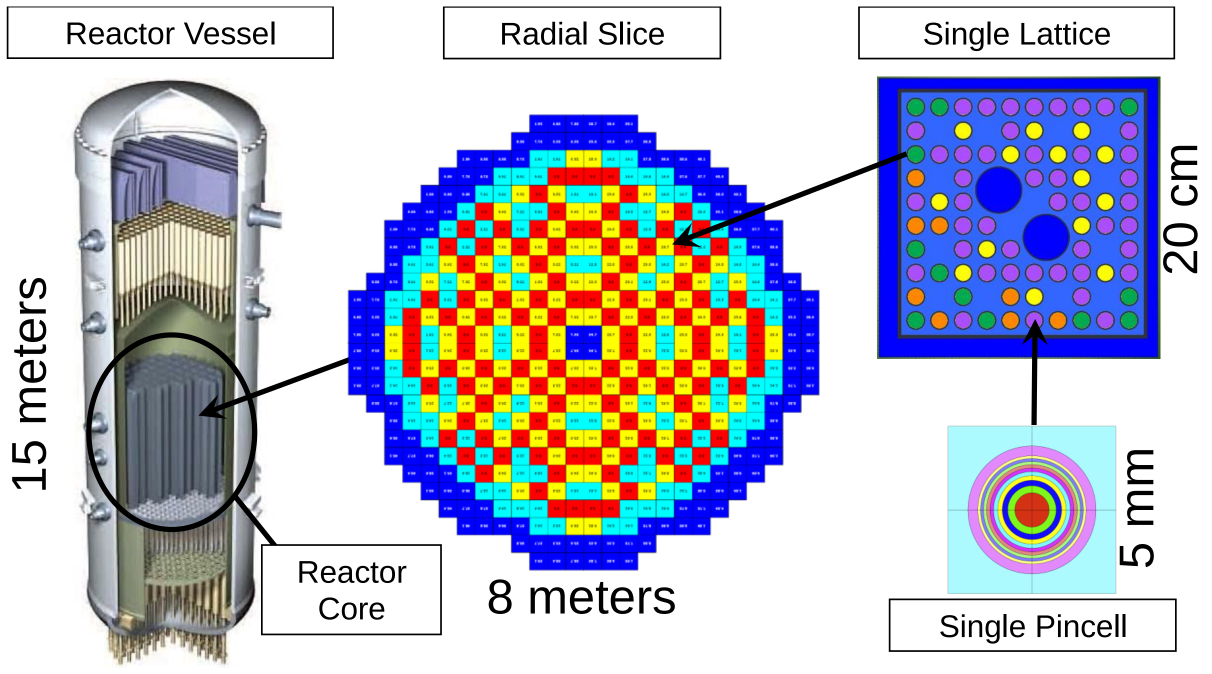

Figure 1 Multi-scale approach

In a pressurized water reactor (PWR), there are approximately 200 fuel assemblies (or bundles) each containing approximately 289 fuel rods (1). This corresponds approximately to 57,800 fuel rods per reactor, which amounts to roughly 6 million 12ft long fuel rods loaded in a US reactor at any given time. There is an estimated average fuel failure rate of 5 rods per million (2), so with an average core residence time of six years (PWR), this translates into an average of five fuel failures per year. In many cases, failed fuel has to be removed from the core, which corresponds to an unscheduled outage and loss of generating capacity for a utility. So if a reactor has to be shut down for a typical three to five days to replace or reconstitute failed fuel, this easily translates into financial losses that exceed several million dollars for even the shortest of shutdowns and restarts. The impact upon operation and safety cannot be measured in terms of dollars, but let it suffice to say that fuel failures cause significant impact and uncertainty upon the day to day operations of a nuclear power plant.

Of fuel failures, approximately 70% are due to grid to rod fretting (3). This is a complex tightly coupled time-dependent phenomenon by which the mechanisms (dimples and springs) holding a rod in place weaken, allowing the rod to vibrate. These flow-induced vibrations wear away at the rod over time until a hole is punched in the rod. The next most frequent fuel failure mechanism is pellet-clad interaction (3), also known as PCI, which accounts for 15% of fuel failures. This is another tightly coupled multi-physics phenomenon by which the pellet comes into thermo-chemical-mechanical contact with the clad, causing a weakening or cracking of the cladding. Of the remaining 15% of fuel failures, the largest percentage is unknown. In some cases, the failure was not investigated, but in many cases the cause could not be determined.

In order to calculate thermo-mechanical response, fuel performance codes such as FRAPTRAN (4) can be used. These codes are extremely sensitive to power, primarily due to its effect upon temperature, which in turn has a positive feedback mechanism through fuel physics phenomena, such as fission gas release. Fission gas release occurs when gaseous byproducts from fission (primarily xenon) are released from the fuel pellet into the gap between the pellet and the clad, which is originally filled with a pressurized gas (typically helium). Xenon has a considerably lower thermal conductivity than helium does, which increases the temperature difference across the gap. A slight increase in power causes a temperature increase, which increases fission gas release, which further increases temperature. Therefore, an accurate spatial and time-dependent prediction of power is essential to correctly modeling most other coupled physical processes affecting fuel performance (5).

As plant capacity factors in the US are now routinely exceeding 90% and new sources of nuclear power are slow to arrive, the demand for additional nuclear power generation is currently being met by a combination of plant life extensions, maximizing irradiation (burnup) of fuel, and through power uprates. This translates into more aggressive fuel and core designs that are likely to push the operational envelope of failure mechanisms.

1.1 MOTIVATION

The motivation for the development of a computationally efficient kinetics method is the practical implementation of the spatial kinetics method within modern, massively parallel, three-dimensional multi-physics packages which include solutions of neutron transport and nuclear fuel performance, primarily thermo-mechanical response of fuel elements. There are two primary kinetics methods in use, which are quasi-static kinetics and direct kinetics, described and derived in Chapter 3. The basis of the quasi-static method is a pseudo separation of variables where the time dependent angular flux is separated into a weakly time-dependent spatial component and a strongly time-dependent, spatially independent amplitude function. Quasi-static kinetics is typically fast and accurate for transients in which the flux shape changes slowly, but can in fact be slower than direct kinetics for transients where the shape changes rapidly (6). The basis of direct kinetics is to solve the precursor and flux equations simultaneously, assuming a finite difference approximation of the temporal derivative of both the flux and the precursor concentrations. The theta method is then applied to the equations in order to discretize time. Fully implicit direct kinetics is the most accurate kinetics solution (7), but requires the solution of a much larger system of equations and very fine time steps. Therefore, hybrid kinetics methods sharing the quasi-static analytic representation of the precursors as well as the theta solution method for the flux equations have been developed in previous works, such as the implementation used by PARCS (8).

A new, simplified hybrid method is developed in this dissertation for the purpose of improved computational time that can still be highly accurate for rapid shape changes, where weighted methods are typically slower (6). This new hybrid method is a modification to a previous diffusion developed kinetics method (9) which will allow the realization of the full potential of the multi-physics framework as well as the utilization of a time-dependent three-dimensional many neutron energy group approach that is still computationally efficient for problems with rapidly changing shapes. The semi-implicit direct kinetics method (SIDK) is also implemented as the higher order backward differentiation formulae (BDF) method presented in (9) if larger time step sizes are required for problems without thermal feedback, since the SIDK method is shown to outperform the higher order backward discretization (HOBD) method for these problems in terms of accuracy, as a single time step for a full fuel assembly coupled neutron transport – fuel performance solution using a moderate amount of neutron energy groups (forty-four) took 400,000 CPU hours on the Jaguar supercomputer at Oak Ridge National Laboratory (10). Thus, computational savings of the SIDK hybrid method versus direct kinetics methods, which ranged from a factor of two to an order of magnitude, are expected to be highly important for the largest computational problems. In particular, it should be noted that the computational savings of an order of magnitude occurred for problems with six neutron precursor groups, where accuracy increases with an increasing number of neutron precursor groups.

1.2 ORGANIZATION OF THIS DISSERTATION

This dissertation is divided into seven chapters, including this introductory chapter, which provides an overview of nuclear reactor simulation and the motivation for this work in the preceding section. Following this section is a description of the computer codes used in this work, the fundamental equations to be solved, and biases introduced by current practice implementations. Chapter 2 consists of a detailed literature review of current multi-physics modeling efforts, time integration strategies, and core simulators. Chapter 3 outlines the derivation of the two-primary kinetics methods in use today, which are the factorization method and the direct (or theta) method, and also provides the derivation of the Semi Implicit Direct Kinetics methodology herein introduced as well as the HOBD method implemented in a two-group diffusion framework (9).

Chapter 4 through Chapter 6 contains the bulk of results presented in this dissertation. Chapter 4 focuses upon point kinetics. This chapter includes the point kinetics equations with and without thermal feedback as well as a description of codes employed. Following this is the development of kinetics parameters, including β and λ values for non-benchmark problems that need this information. Finally, point kinetics results with and without thermal feedback are presented. Chapter 5 and Chapter 6 focus upon spatial kinetics results, primarily presenting a variety of benchmark problems as well as a "demonstration of capability" problem. These benchmarks consist of:

- Infinite Homogeneous Medium (IHM) (11) - An infinite homogeneous medium problem to demonstrate linear convergence to an analytic (exact) solution.
- 16-A1 (12) (11) - A one-dimensional fast reactor transient without feedback to demonstrate speed and accuracy of the method versus the direct method for diffusion and transport implementations.
- TWIGL (13) (6) - A two-dimensional seed blanket reactor with two transients to demonstrate the speed and accuracy of the method versus a number of other fully-coupled time integration strategies.
- LRA (6) (12) (14) - A two-dimensional prompt-supercritical transient with feedback to demonstrate the speed and accuracy of the method versus quasi-static (factorization), direct and HOBD methods.

While the "demonstration of capability" problem is:

- Fully-Heterogeneous Mini-Assembly (FHMA) - A three-dimensional transport based, forty-four energy group control rod movement that is compared to point kinetics to demonstrate the capabilities of the SIDK method and the AMP-Denovo framework developed.

Within these problems, the following comments are provided on discretization of time, neutron energy, angle and space which is the focus of this dissertation:

- IHM – Discretization of time only, as there is no spatial or angular dependence, and a two-group neutron energy structure is used.
- 16-A1 – Discretization of space, time and angle are all examined in this problem as there is time, space and angular dependence. Two-group neutron energy structure used.
- TWIGL – Discretization of time and space, neutron diffusion with two-group neutron energy structure.
- LRA – Discretization of time and space in the presence of thermal feedback, neutron diffusion with two-group neutron energy structure.
- FHMA – Discretization of time, space, angle, and energy as all are finely treated in the fully-3d radiation transport calculation utilizing fine angle and neutron energy treatment.

Lastly, Chapter 7 presents the conclusions of this dissertation. The conclusions include a summary of relevant findings as well as closing comments including proposed future research.

1.3 UNDERLYING FRAMEWORK FOR THIS RESEARCH

Of primary focus in this dissertation is the development of space-time neutron kinetics models that can readily interface within the multi-physics framework Advanced Multi-Physics (AMP) (15) (16) (17) (18) which present a stable choice that is a decent starting point for large scale, parallel simulations of multi-physics neutron kinetics problems with thermal feedback. This dissertation illustrates the ranges of problems that the theta method with an analytical representation of precursors, denoted as a “hybrid method” in this dissertation, can be expected to work well for in parallel, neutron transport, multi-physics implementations. The novel components of this dissertation include the refinement of hybrid kinetics methods within a neutron transport framework, the implementation of fine neutron energy discretization in the time-dependent Boltzmann transport solutions, and the application of supercomputing environments to the solution of the time-dependent Boltzmann equation (19) (10). This work is further supported by an in-depth study of various space, time, and angle discretization strategies, some of which have been studied or compared in previous works (6) (13) (9), among others.

The neutronics analysis capabilities are herein provided by the Denovo (20) radiation transport code. In order to develop and flexibly test the hybrid methodology, an independent two-group time-dependent "surrogate" neutron diffusion code was written which is used for problems with and without thermal feedback (this is denoted as the SIDK program, named for the SIDK method). Time integration analysis for every problem is provided by codes written with the MATLAB (21) program, which has proven to be a robust point and spatial kinetics solver for problems with and without thermal feedback. The point kinetics and spatial kinetics amplitude function calculations come from the Burner Reactor Integrated Safety Code (BRISC) (22), which is integrated with the multi-physics framework, the AMP code (17).

1.3.1 DESCRIPTION OF COMPUTER CODES

A bulleted list of the codes that were used or modified is shown below, with key citations noted and followed by brief descriptions of these codes.

- AMP (15) – Advanced Multi-Physics Nuclear Fuel Integrated Performance and Safety Code. Used as multi-physics framework.
- BRISC (22) – Point and spatial kinetics package incorporated within AMP. Being used for point kinetics calculations.
- MATLAB (23) (21)– Generic mathematical solution package. Utilized for point and spatial kinetics calculations utilizing a variety of time integration techniques.

- SIDK (11) – Time-dependent two-group neutron diffusion code written for this dissertation based on the SIDK method, developed in this dissertation. Used for investigation of comparison with fully implicit direct kinetics with and without thermal feedback, as well as time integration strategies.
- CONQUEST (6) – Nodal three-dimensional time-dependent neutron diffusion code based on quasi-static kinetics implementation. Used as benchmarking for two transients.
- TDTORT (13) – Discrete ordinate three-dimensional time-dependent transport code based on quasi-static kinetics implementation. Used as benchmarking for two transients.
- Denovo (20) – Discrete ordinate three-dimensional steady-state radiation transport code. Adapted for time-dependent calculations in this dissertation via the SIDK method.
- SCALE (24) – Neutronics code suite used for cross section processing and radiation transport.
    - CSAS-I – Cross section collapsing routine used to prepare cross sections for Denovo.
    - TRITON/NEWT – Two-dimensional discrete ordinate radiation transport code being used for development of point kinetics parameters, benchmarking Denovo and development of problem dependent collapsed cross section libraries for CSAS-I.

Neutron transport codes used in this work primarily consist of: NEWT and Denovo, for 2D and 3D applications, respectively. The primary purpose of these codes is to solve the Boltzmann neutron transport equation. All of these codes can solve problems from the pin to the core scale and are often used as lattice-physics codes to feed cross sections for neutron diffusion applications. NEWT and Denovo are both deterministic codes, as opposed to a stochastic code such as MCNP (25). SCALE modules and the MCNP code are both extremely well validated. Denovo has been validated against the NEA’s C5G7 MOX benchmark (26), as well as other problems.

Software used in this work for point kinetics primarily consists of: BRISC, MATLAB, and AMP. The primary use of these codes is to solve the spatially-independent point kinetics equations. Point kinetics equations form a system of ordinary differential equations (ODEs) that must be integrated over time. Time integrators used in this work include: Eigenvalue decomposition (27), Pade (28), Runge-Kutta (29), and numerical differentiation formulas (NDFs) (30). Point kinetics is typically applied at the core-wide scale.

For the solution and evaluation of the time and space-dependent Boltzmann neutron transport equation, software utilized consists of the SIDK time-dependent neutron diffusion program, MATLAB, and the Denovo radiation transport solver herein adapted. Comparison programs consist of TDTORT and CONQUEST. Spatial kinetics is also typically applied at the core-wide scale, which involves solving an eigenvalue problem, perturbing the system, and then solving repeated fixed source problems to advance in time, with the time step size dependent upon the method selected, for reasons of accuracy or numerical stability. Time integration strategies considered in this work include: trapezoidal rule followed by implicit Runge-Kutta (TR-BDF2), numerical differentiation formulas (NDFs), HOBD, SIDK, fully-implicit direct kinetics, and second order SIDK.

1.4 Overview of Fundamental Principles

In brief, a nuclear reactor's primary purpose is to steadily and safely generate full rated power while satisfying reactivity and thermal margins (i.e., operating within licensing limits and plant technical specifications). At a steady "critical" state (zero reactivity) the reactor is generating the same amount of neutrons between successive generations, or as a function of time. These neutrons are produced from fission, a subset of neutron absorption reactions that lead to compound nuclei that spontaneously split into fission fragments (products) and generate two to three additional neutrons. These generally high energy neutrons, if they don't leak or aren't lost to resonance absorptions, are moderated (slowed down) by the hydrogen in the water of a light water reactor, thereby increasing the probability of subsequent fissions, which each releases about 200 MeV of energy, most of it recoverable. These neutrons also activate materials, as well as produce elements heavier than uranium (known as transuranics) through absorptions. The fission process and neutron absorptions create other unstable isotopes that radioactively decay, some by delayed neutron emission. Delayed neutrons, although a small fraction of the total number of neutrons in a reactor (a fraction of about 0.0065 of the total neutron generation in U-235 based fuel), are an important aspect of the controllability of nuclear reactors and constitute the focus of neutron kinetics.

A key equation that describes the balance and spatial and time-dependent characteristics of neutrons is the Boltzmann transport equation (1). This equation governs the transport, absorption, scatter, and production of neutrons. The terms in this equation at a given energy are production and destruction, where production is from fission, in-streaming, in-scattering, or delayed neutron emission while destruction is from absorption, out-scattering, or out-streaming (31). A steady-state version of the Boltzmann neutron transport equation is shown below:

$$\widehat{\Omega} \cdot \nabla\Phi(\vec{r},E,\widehat{\Omega},t) + \Sigma_t(\vec{r},E,\widehat{\Omega},t)\Phi(\vec{r},E,\widehat{\Omega},t) = \int_{4\pi} d\widehat{\Omega}' \int_0^\infty dE'\Sigma_s(\vec{r},E' \rightarrow E,\widehat{\Omega} \rightarrow \widehat{\Omega}',t)\Phi(\vec{r},E',\widehat{\Omega}',t) + \frac{1}{k}\int_{4\pi} d\widehat{\Omega}' \int_0^\infty dE'\nu\Sigma_f(\vec{r},E',\widehat{\Omega}',t)\Phi(\vec{r},E',\widehat{\Omega}',t) + S(\vec{r},E',\widehat{\Omega}',t) \quad \text{Equation 1.1}$$

This equation has seven independent variables. The independent variables are three spatial coordinates (x, y, z), two angles (azimuthal and polar, combined into solid angle), energy, and time. The terms on the left represent angular flux streaming (leakage) and absorption, respectively. The terms on the right represent in-scatter, eigenvalue, production from fission, and external source. Equation 1.1 is typically solved in steady-state form (neglecting temporal derivatives) with a slow-varying time dependence primarily coming from transmutation (depletion/burnup).

The general equation for the concentration of one nuclide N_i can be expressed as the difference between the formation rate and the destruction rate:

$$\frac{dN_i}{dt} = \text{Formation Rate} - \text{Destruction Rate.} \quad \text{Equation 1.2}$$

Any process in which a nuclide N_i is converted to another nuclide (or multiple nuclides) is considered a destruction of that nuclide, i.e. the (n,γ) reaction ${}^{b}_{a}N + {}^{1}_{0}n \rightarrow {}^{b+1}_{a}N + \gamma$. Any process in which another nuclide is converted to N_i is considered a formation, i.e. ${}^{b-1}_{a}N + {}^{1}_{0}n \rightarrow {}^{b}_{a}N + \gamma$. There are a multitude of reactions which can take place under the presence of a high neutron flux, including (n,2n), (n,p), (n,γ), (n,f), etc. However, the rates of change of nuclide concentrations are generally dominated by the (n,γ) and (n,f) reactions, as well as radioactive decay. The result is the time-dependent nuclide concentration equation. Equation 1.3 below characterizes the rate of change of the number density for an arbitrary isotope i based on gains from radioactive disintegrations from isotopes j via branching fractions l_{ij}, gains from neutron absorptions from isotopes k via branching fractions f_{ik}, and losses from disintegration, λ_i, and neutron absorptions, σ_i, of isotope i.

$$\frac{dN_i}{dt} = \sum_{j=1}^{m} l_{ij}\lambda_j N_j + \bar{\Phi}\sum_{k=1}^{m} f_{ik}\,\sigma_k N_k - (\lambda_i + \bar{\Phi}\sigma_i)N_i \qquad \text{Equation 1.3}$$

where the summation over j encompasses all nuclides N_j which may produce N_i as a result of radioactive disintegrations, and the summation over k encompasses all nuclides N_k which may produce N_i as a result of neutron absorptions . The other terms account for the destruction of N_i via neutron capture or decay. The neutron flux Φ is a function of the neutron energy, as is the effective cross section σ. The key to solving this equation for every nuclide N_i is obtaining accurate estimates of the neutron flux and effective cross sections as a function of neutron energy, which comes from the solution of Equation 1.1. (32)

An accurate and detailed depiction of the power distribution is essential for thermo-mechanical response calculations, which are provided by fuel performance simulations, whereby power is directly related to the neutron flux density and, more specifically, to the fission reaction rate density distribution. However, predicting the actual power distribution is quite challenging because it requires a global (core-wide) assessment (typically handled by a decoupled neutronics code), while fuel performance evaluations need very localized inputs at the pin or even within-pellet level. Therefore, in practice, feedbacks such as fission gas release on the fuel pin scale are only loosely incorporated, as well as a coarse temperature feedback within core physics codes. These difficulties are further compounded in time-dependent problems because there are global effects (i.e., rod insertions) as well as local effects (e.g., radial power profiles from steady-state are not necessarily appropriate when the temperature distribution in the pin deviates from a standard quadratic shape). Therefore, accurate temperature feedback treatment within transient simulations is paramount to achieving high fidelity in predictive nuclear fuel performance evaluations. This accurate treatment necessitates the development of a high-fidelity radiation transport solver utilizing a computationally efficient and accurate method which is coupled to a fuel performance code, which is one of the objectives of this dissertation. The radiation transport solver used is Denovo, the fuel performance and multi-physics framework that is coupled with Denovo is AMP (10), and the method developed in the dissertation is referred to as the SIDK method, which is demonstrated as an accurate and efficient approach, and it constitutes a new space-time kinetics direct method with an analytic representation of neutron precursors that has been implemented within the framework of the neutron transport equation.

An overview of a typical transient calculation by a core simulator is shown in Figure 2 below. In this diagram, Step 1 constitutes a typical cross section generation process for a simulator that leads to a steady state criticality search calculation. Step 2 postulates a reactivity insertion that perturbs the critical condition leading to a point kinetics time-dependent calculation. The outcome of Step 2 leads into the re-evaluation of the temperature distribution based upon the changes in the power distribution, which constitutes Step 3. Although Figure 2 does not show this, the output from Step 3 would effectively provide the inputs needed by a fuel performance code, which is where a designer would evaluate the fuel’s thermo-mechanical performance based upon operational or design conditions.

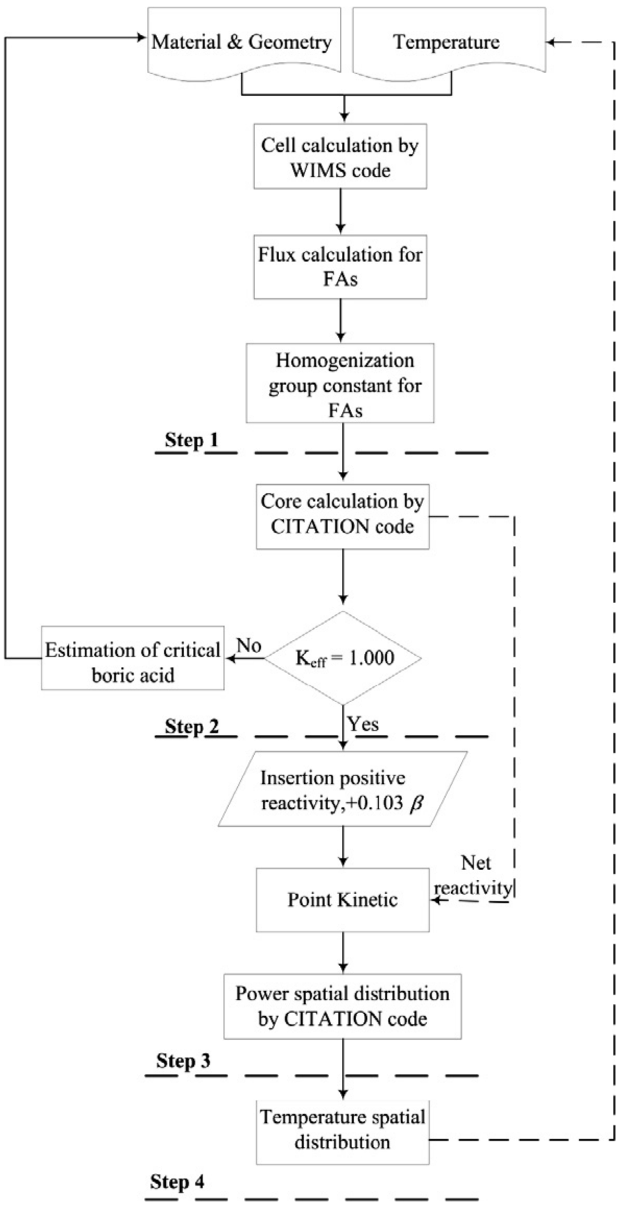
Material & Geometry
Temperature
Cell calculation by WIMS code
Flux calculation for FAs
Homogenization group constant for FAs
Step 1
Core calculation by CITATION code
Estimation of critical boric acid
No
K_eff = 1.000
Yes
Step 2
Insertion positive reactivity,+0.103 β
Net reactivity
Point Kinetic
Power spatial distribution by CITATION code
Step 3
Temperature spatial distribution
Step 4

Figure 2 Typical Transient Core Simulator Process (33)

In order to account for time-dependent changes in reactor operation, the time-dependent form of the Boltzmann transport equation must be used which is presented as Equation 1.4, below:

$$\frac{1}{v}\frac{\partial\Phi}{\partial t}(\vec{r},E,\widehat{\Omega},t)+\ \widehat{\Omega}\cdot\nabla\Phi(\vec{r},E,\widehat{\Omega},t)+\ \Sigma_t(\vec{r},E,\widehat{\Omega},t)\Phi(\vec{r},E,\widehat{\Omega},t)=\int_{4\pi}d\widehat{\Omega}'\int_0^\infty dE'\Sigma_s(\vec{r},E'\rightarrow E,\widehat{\Omega}\rightarrow\widehat{\Omega}',t)\Phi(\vec{r},E',\widehat{\Omega}',t)+\chi_p(r,E)(1-\beta)\int_{4\pi}d\widehat{\Omega}'\int_0^\infty dE'\nu(r,E')\Sigma_f(r,E',t)\Phi(\vec{r},E,\widehat{\Omega},t)+\sum_{i=1}^{I}\chi_i^D\lambda_i C_i(r,t)+S\left(\vec{r},E',\widehat{\Omega}',t\right)\qquad\text{Equation 1.4}$$

where υ is velocity and t is time. Introducing time-dependence further complicates the solution of the transport equation, as the vast majority of neutrons (over 99%) are “prompt neutrons” and are generated extremely fast (within less than 1 ms), while the generation time of the remainder of neutrons is quite long (8 to 12 s). These more slowly released neutrons come from radioactive decay of isotopes that decay by neutron emission and are called “delayed neutrons,” while their parent isotopes are called delayed neutron precursors. This allows the fission source to be split into the prompt and delayed fission neutrons, which is an important aspect of reactor kinetics, and an aspect of traditional neutronics that is generally ignored within fuel performance codes.

Given the lack of use of transport calculations within fuel performance codes, due to computational limitations, the neutron kinetics evaluations are typically limited to space independent or “point” kinetics. However, even this lower-level of fidelity in the treatment of neutron kinetics is not present in any fuel performance code, point or otherwise (34) (35). Likewise, from the literature reviewed, the treatment of thermal responses or feedback also appears to be absent within the realm of fuel performance evaluations; in particular, the evaluation of spatially dependent thermal feedback that results from coupling transport theory to spatial neutron kinetics to generate space-time-dependent power distributions at the pin level scale.

Contemporary fuel performance codes typically employ user-provided power inputs, which are often calculated by external core simulators (such as PARCS (8)). However, as described by Hursin et al (36), these analyses generally smear entire assemblies into homogenized regions, both for thermal feedback and for power calculation purposes. Therefore, the pin power reconstruction data (which relied on assembly averaged temperatures) are missing a significant amount of local information, which is being relied upon heavily by the time-dependent fuel performance codes. As mentioned previously, fuel performance codes (and their accompanying results) are extremely sensitive to power inputs. In addition to this, the simplifications typified in steady-state fuel performance codes (such as one-dimensional radial representation of within-pin powers) is carried through to time-dependent fuel performance codes as well.

These factors comprise the motivation for the development within this dissertation of a computationally efficient and accurate time-dependent neutron transport method that is generically applicable as well as coupled with a fuel performance code in a parallel multi-physics framework (37) (10). This compounding level of smearing and simplifying in current approaches could potentially limit the understanding of space-time-dependent power-temperature distributions, which are vital to calculating material performance for a design basis or beyond design basis accident. This type of information or lack thereof, can impact fuel failure mitigation (through design), can affect core licensing applications, and has had a measurable impact upon limiting or denying power uprate applications (4) (38).

Chapter 2 LITERATURE REVIEW

The DOE Office of Nuclear Energy (DOE/NE) established an Advanced Modeling and Simulation Office (AMSO) for the development of predictive simulations tools that will develop integrated performance and safety codes for the predictive modeling of key nuclear energy problems. The concept of predictive simulation is to remove (or minimize) the use of experiments to define simulation tools by incorporating the first-principle effects of the physics and upscaling them to the traditional "engineering" scale. There are currently two parallel efforts– the Consortium for Advanced Simulation of LWRs (CASL) (39), focusing primarily upon Pressurized Water Reactors (PWRs), and the Nuclear Energy office of Advanced Modeling and Simulation (NEAMS) (40), which is developing tools that can be applied to a wide variety of applications. In the European Union, there is a similar effort (F-BRIDGE) (41) for advancing nuclear software to better leverage advanced computing platforms and improve the predictability of the software.

There are several individual physics components (neutronics, fuel physics, flow, mechanics, chemistry, etc.) associated with each of these programs and each leverages some form of a computational backplane to integrate the physics into a useable form for the engineer analyst. These include AMP (17), Bison/Marmot (42), and Pleadies (43), for fuel performance, and Denovo (20), MPACT (44), and DeCart (45) for radiation transport, which are built upon various (similar, but different) platforms, including LibMesh (46), MOOSE (42), LIME (22), SHARP (47), and MOAB (47). Each of these programs is built, to a degree, on the knowledge that there are significant aspects that are being neglected by the traditional tools, which limits their ability to be predictive to within only their empirically derived bounds. One example of this is a traditional empirical limitation in fuel performance codes, which are validated for uranium-dioxide (UO_2) fuel under 5% enrichment, where enrichment refers to the weight percent of the uranium that is ^{235}U.

2.1 RELEVANCE OF PREDICTIVE SIMULATION TOOLS

The research described in this dissertation aims toward a higher level of resolution by employing a new approach to kinetics, denoted the semi-implicit direct kinetics (SIDK) method. This approach is demonstrated to be robust, accurate, faster executing than traditional direct kinetics by roughly an order of magnitude and amenable to high performance computing, thus, capable of producing high resolution neutronics for fuel performance applications. The key improvement metric of the SIDK method over all other methods considered in this work, including the HOBD and other higher order methods, is the error versus run time ratio, in contrast to the higher order methods for suitably high error tolerances (0.1 % relative error). In addition, the SIDK method exhibits improved stability specifically on problems with thermal feedback when contrasted against traditional direct kinetics and the HOBD method.

One of the key areas of focus is the application of the SIDK methodology to interface with the AMP-Denovo framework for transient applications. There were two specific tasks completed to facilitate this objective; first, to integrate an existing kinetics package within AMP, verify the package, and to couple this package within AMP's multi-physics canvas in order to provide spatial thermal feedback to the neutron kinetics (Chapter 4). This development included the development of kinetics parameters from transport solutions as well as integration with Denovo (20). The second task implemented a spatial kinetics capability within Denovo using the new semi-implicit direct kinetics method (Chapter 6). This work is compared to reactor level simulations for verification (Chapter 5), and then applied to fuel pin level simulations (Chapter 6).

2.2 TIME INTEGRATION

Traditionally, reactor kinetics is applied as a time-stepping scheme for the space-energy discretized Boltzmann neutron transport equation shown in Equation 1.4, which represents a splitting of the space-time dependence of the equation. This approach is consistent with that used in other fields, such as in the advection-diffusion problem (48). As such, the primary purpose of reactor kinetics is to develop a computationally efficient and accurate temporal representation of Equation 1.4. Therefore, we will start the kinetics literature review by examining various temporal solution methods, both applied to kinetics problems and general problems.

There are three basic approaches to time integration. These approaches consists of explicit methods, semi-implicit methods, and fully implicit methods, where explicit refers to using the evaluation of the function at the previous time step to advance the solution, implicit refers to using the evaluation of the function at the current time step, and semi-implicit is some combination thereof. To illustrate this, we will examine a common semi-implicit method, which is the Crank-Nicholson method, a finite difference method that is of second order in time and is numerically stable (8). The Crank-Nicholson method is effectively the average of the solution to the differential equation set at time t and time t+1, or basically a weighted average of the forward Euler and backward Euler methods, or the fully explicit and fully implicit methods. It should be noted that when using an explicit or unstable semi-implicit approach, numerical instabilities can lead to oscillations in the solution process if the ratio of time step size to the dimensional mesh area (i.e., $\Delta t/\Delta x^2$) is larger than around one half (49). The equations below illustrate the flux dependence on time in the above-noted approaches: (28)

$$\frac{\partial\phi}{\partial t} = F(\phi, t, \frac{\partial\phi}{\partial t}, \frac{\partial^2\phi}{\partial t^2})$$ Flux Dependence on time Equation 2.1

$$\frac{\phi_i^{t+1} - \phi_i^t}{\Delta t} = F_i^t(\phi, t, \frac{\partial\phi}{\partial t}, \frac{\partial^2\phi}{\partial t^2})$$ Forward-Euler (explicit) Equation 2.2

$$\frac{\phi_i^{t+1} - \phi_i^t}{\Delta t} = F_i^{t+1}(\phi, t, \frac{\partial\phi}{\partial t}, \frac{\partial^2\phi}{\partial t^2})$$ Backward-Euler (implicit) Equation 2.3

$$\frac{\phi_i^{t+1} - \phi_i^t}{\Delta t} = \frac{1}{2}(F_i^{t+1}(\phi, t, \frac{\partial\phi}{\partial t}, \frac{\partial^2\phi}{\partial t^2}) + F_i^t(\phi, t, \frac{\partial\phi}{\partial t}, \frac{\partial^2\phi}{\partial t^2}))$$ Crank-Nicholson (semi-implicit) Equation 2.4

2.2.1 MATRIX EXPONENTIAL METHODS

Another solution method for time integration, which is available in the BRISC package, is a Pade approximant (28) (27), which is basically the best approximation of a function by a rational function of a given order. The default BRISC order for the Pade solver is of third-order. The Pade approximant will usually give a better approximation of a function than its truncated Taylor series and may also offer a solution when a function does not have a convergent Taylor series. So, for example, an arbitrary function F, which depends on flux, time, and first and second partial derivative of the flux with respect to time, is given by Equation 2.5.

$$F = F\left(\phi, t, \frac{\partial \phi}{\partial t}, \frac{\partial^2 \phi}{\partial t}\right) \qquad \text{Equation 2.5}$$

Thus, the Pade approximant of this function of order (m,n) is given as the rational function which is:

$$R(x) = \frac{p_0 + p_1 x + p_2 x^2 + \cdots + p_m x^m}{1 + q_1 x + q_2 x^2 + \cdots + q_n x^n} \quad \text{Pade Approximant (28)} \qquad \text{Equation 2.6}$$

Where R is the evaluation of the function and p and q are coefficients.

Alternatively, for a matrix that is fixed in time (linear), an eigenvalue decomposition can be used. For the infinite homogeneous medium problem that is presented in section 6.1 , this is the implementation that was used, as it is one of the few ways to obtain an exact solution to a multi-group kinetics problem (27). Starting from the time-dependent Boltzmann equation with explicit representation of delayed neutrons given by Equation 1.4 and the change in precursor concentrations given by:

$$\frac{\partial C_i}{dt} = \beta_i F D \psi - \lambda_i C_i \qquad \text{Equation 2.7}$$

We can form a matrix A such that

$$P(t) = e^{At}, \qquad \text{Equation 2.8}$$

which is exact, as long as A is not changing with time, where A is given by a combination of Equation 1.4 and 2.7 yielding:

$$A = \begin{bmatrix} [L - MSD - (1-\beta)MFD] & \cdots & \chi_i\lambda_i \\ \vdots & \ddots & \vdots \\ \beta_i FD & \cdots & -\lambda_i \end{bmatrix} \qquad \text{Equation 2.9}$$

As can be seen, the only difference between these equations and the traditional point kinetics equations is the discretization of powers by neutron energy group to allow for the explicit representation of the mean generation time of each energy group. The net rate of power change is computed by summing the individual power changes. There are two common solution approaches employed in the point kinetics solutions, which are matrix exponential methods (such as Pade) and Runge-Kutta methods (such as backward Euler, RK4, RK45, TR-BDF2, etc.). Two such methods were developed are benchmarked in the point kinetics section 4.3 . While the Runge-Kutta methods are typically quite fast, they can become unstable for stiff systems, such as fast reactor transients, which can necessitate such small time steps that the method is no longer fast. Pade iterative matrix methods are generally slow, but highly accurate for such systems, depending upon the order of Pade selected. The orders of convergence of both methods were explored. Alternatively, an eigenvalue decomposition can be used, which it is in section 6.1 such that the exponential A matrix is decomposed into $A=XDX^{-1}$ where X is composed by the eigenfunctions of the system (27).

2.2.2 MULTI-STAGE METHODS

In order to verify the BRISC kinetics package prior to and following its integration into AMP, independent MATLAB programs were written using a multi-step adaptive Runge-Kutta fourth and fifth order method for testing and comparison. This was done using a built in time integration scheme for coupled ODEs that is available in MATLAB. The time dependence is approximated using a number of sub-steps (or stages), and the error is approximated by using a Runge-Kutta of an order one higher than the methods number of sub-steps. This is advantageous in that few additional calculations per time step need to be done to approximate the error with a higher order Runge-Kutta method. The general formulation for a Runge-Kutta method is (28):

$$y^*_{n+1} = y_n + \sum_{i=1}^{s} b_i^* k_i, \qquad \text{Equation 2.10}$$

where y is a function, n denotes the time index, i is an integer, b is a coefficient, k is the value of the function evaluated at a sub-interval, and s is the order of the method. Here, the set $\{k_i\}$ for i=1,s is the same as for the higher order method. The error is then given by:

$$e_{n+1} = y_{n+1} - y^*_{n+1} = h \sum_{i=1}^{s} (b_i - b_i^*) k_i. \qquad \text{Equation 2.11}$$

Where e is the error and h is the time step size. The RK4 method, specifically, is given by:

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4), \qquad \text{Equation 2.12}$$

$$k_1 = f(t_n, y_n), \qquad \text{Equation 2.13}$$

$$k_2 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} k_1\right), \qquad \text{Equation 2.14}$$

$$k_3 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} k_2\right), \qquad \text{Equation 2.15}$$

$$k_4 = f(t_n + \Delta t, y_n + \Delta t k_3) \quad . \qquad \text{Equation 2.16}$$

Where t is time. Other multi-stage methods utilized in this work included TR-BDF2 which is a trapezoidal rule followed by a second order Runge-Kutta solution of order two. While the Runge-Kutta is a multi-stage method, as it requires multiple calculations per time step solution, the TR-BDF2 is also a multi-step method, as it uses information from more than just the previous time step. Multi-step methods are covered in the following sub-section, 2.2.3 .

2.2.3 MULTI-STEP METHODS

In order to improve the convergence and time step size of implicit or explicit methods, an alternative to the multi-stage methods are multi-step methods. These methods offer higher order of convergence with no additional computational cost by using several previous solutions in the current solution, which does add a minimal amount of increased storage. These methods are typically the most useful when very high accuracy is desired (50). The general backward method for time dependent change in function *U* can be written as:

$$U(t_{j+1}) + \alpha_1 U(t) + \alpha_2 U(t_{j-1}) + \cdots + \alpha_m U(t_{j+1-m}) = hc_o f\left(t_{j+1}, U(t_{j+1})\right), \qquad \text{Equation 2.17}$$

where *c* and α are coefficients whose values can be found in Ginestar (9) as well as many other references. Multi-step methods considered in this work include: HOBD (order 2), SIDK (order 2), TR-BDF2 (order 2), and NDF (variable order).

2.3 REACTOR TRANSPORT AND KINETICS CODES

Modeling a nuclear reactor in great detail and with high fidelity ideally requires an integrated multi-physics approach consisting of a full neutronics analysis coupled to a full thermo-mechanical-flow analysis, potentially also incorporating chemical effects. The primary physics for the neutronics typically include neutron transport methods which can be modeled by modules within the SCALE code suite or also by stochastic codes such as MCNP. Isotopic depletion, transmutation, and decay can be handled in a number of ways, including by simplified macroscopic depletion models or by direct coupling to well-known microscopic depletion codes such as ORIGEN (24).

A quasi-static approach is used for standard operation, with depletion (Eq. 1.2) used to calculate the changing isotopics for each large time step. The next subset of physics typically modeled is the thermo-mechanical response, which uses quasi-static powers generated from the neutronics analysis. Examples of fuel performance codes include FRAPCON (5), FALCON (35), TRANSURANUS (51), Bison, and AMP. Historically, neutronics codes have placed a primary focus upon thermal-hydraulic response, as this is the primary mechanism that affects the neutron cross-sections via temperature feedback. Codes that calculate this thermal-hydraulic response include RELAP (52), STAR-CD (36), or COBRA (53).

For transient analysis, it is more difficult to decouple the inherently coupled feedback of thermo-mechanics from spatial dependent reactor kinetics, for which three main implementations are described in the literature for this coupling (8) (36) (54) (33). The first implementation is to totally decouple the thermo-mechanical-flow analysis, and use spatially dependent reactor kinetics to predict power changes in a reactor. These power changes can then be fed on a pin-by-pin basis to thermo-mechanics codes to calculate the thermo-mechanical response, with the idea that the spatial flux distribution is sufficiently decoupled from thermal response to make this a realistic approach. (13)

The second approach is to couple the thermo-mechanical response with the spatially independent (point) kinetics equations. This approach essentially adds a number of ordinary differential equations (ODEs) to the point kinetics system, and solves the entire system with no spatial dependence, or lumped spatial dependence (55). The heat equation is applied to entire regions (fuel, clad, coolant, control banks), with one equation for each region. A related approach entails using lumped capacitance models for each region in conjunction with the point kinetics equations. These results can be fed to a quasi-static flux calculation (33). The third approach is to couple the spatial kinetics to spatial thermal-hydraulic calculations, which is done in core simulators such as NESTLE, PARCS, and DeCart with Star-CD (36).

The BRISC (22) kinetics package is an example of a point kinetics package, which has been implemented within AMP. These calculations require a solution of the "shape" or transport function. One approach is to couple the kinetics equations using diffusion theory, which is similar to how kinetics are handled in the NESTLE nodal simulator. Both a diffusion solution and a kinetics solution were run as part of the integration with AMP. In BRISC, the 2D diffusion package RASCAL (56) is used to solve the shape function. Following is a brief overview of common methods and equations used in both steady-state and time-dependent diffusion and transport approaches to kinetics and coupled calculations.

2.3.1 STEADY-STATE FLUX CALCULATIONS

There are several ways to model neutron transport including diffusion, discrete ordinates, method of characteristics, Monte Carlo statistical methods, and integral transport, to name some of the most common. Method of Characteristics (MOC) and integral transport are described in (45) while the Monte Carlo technique is described in (25). Of these approaches, diffusion theory is typically the fastest and simplest, thus, described first. The multi-group diffusion theory eigenvalue equation is described as follows:

$$-\nabla \cdot D_g(r)\nabla\phi_g(r) + \Sigma_{rg}\phi_g(r) = \sum_{g'=1}^{g-1} \Sigma_{sg'g}(r)\phi_g(r) + \frac{1}{k}\chi_g \sum_{g'=1}^{G} \nu_{g'}\Sigma_{fg'}(r)\phi_g(r) \quad \text{Equation 2.18}$$

where D is the diffusion coefficient, g is the energy group, ϕ is flux, Σ_{Rg} is the removal cross section, k is the multiplication factor (inverse of the eigenvalue), χ_g is the fission spectrum, Σ_{Sg} is a scattering cross section, G is number of energy groups, ν_g is neutrons per fission, and Σ_{fg} is fission cross section (1). One common discretization of these equations is via the finite difference approximation, by which one approximates derivatives by using a first-order Taylor series estimate based on functional values at discrete "nodes" which then become unknowns within a coupled linear system of equations.

$$f'(x) = \frac{f(x+h)-f(x)}{h} \quad \text{Equation 2.19}$$

NESTLE (54) is an example of a code that uses a finite difference discretization but which employs what is referred to be a "nodal" method. Nodal methods effectively employ higher order estimates to the derivatives or gradients, thus, are more accurate at capturing neutron leakage across coarse-mesh or larger nodes (assembly or bundle size nodes). The cross sections for diffusion or nodal simulators are provided by transport based lattice physics codes such as CASMO (57) or TRITON, as with the BRISC package. The numerical version of Equation 2.18 effectively becomes a linear system of equations described by the following eigenvalue and eigenfunction system of equations:

$$\bar{\bar{A}}\vec{\phi} = \frac{1}{k}\bar{\bar{B}}\vec{\phi} \quad \text{Equation 2.20}$$

where *A* is the diffusion matrix, ϕ is the flux vector, k is the multiplication factor, and *B* is the source matrix. Alternatively, the flux and other properties can preserve their angular dependency by solving the Boltzmann transport equation. One approach to solve this equation is the discrete ordinates approach, which is the primary approach considered in this work and which discretizes the continuous variable of angle into discrete angles. At these discrete angles, integrated values of angular flux are calculated. These discrete angular fluxes can be converted to scalar fluxes using a weighted sum via quadratures in order to calculate angle independent reaction rates as described below.

$$\phi(r,E) = \int_0^{4\pi} \psi(r,E,\Omega)d\Omega = \sum_{n=1}^{N} \omega_n \psi(r,E,\Omega_n), \qquad \text{Equation 2.21}$$

where ϕ is scalar flux, ψ is angular flux, Ω is solid angle, and ω is quadrature weight. In order to calculate the flux distribution, a discrete ordinates code such as Denovo (20) can be used. Denovo can solve the k-eigenvalue as well as the fixed source formulation of the transport equation, both of which are heavily utilized in this work. The k-eigenvalue form of the Boltzmann transport equation is described by Equation 1.1.

2.3.2 TRANSIENT FLUX CALCULATIONS

In order to calculate time-dependent flux distributions, a simplification that was previously made in Equation 1.1 neglecting the temporal derivative of the flux must be removed, as it was in Equation 1.4. Furthermore, as the generation time of the vast majority (over 99%) of “prompt” neutrons is extremely fast (less than 1 ms), while the generation time of the remainder of “delayed” neutrons is quite long (8-12 s), the fission source must be split into the prompt and delayed fission neutrons. This approach to treating the fission neutrons on the basis of their generation time forms the basis for the reactor kinetics equations, which are based upon accounting for the concentrations of delayed neutron precursors, so to adjust the source term. By modeling their concentrations, the reactor’s time-dependent behavior can be determined with a fair degree of accuracy, even if using 0D models (22).

The above-noted 0D models are also known as the "point kinetics equations" and are effectively the same equation set as Equations 1.4 and 2.7, but which neglects spatial dependencies, and in this case, also energy dependence. Point kinetics are used for verification purposes as well as comparison with analytical solutions.

If one uses the multi-group diffusion equations (54), the time-dependent form becomes

$$\frac{1}{v_g}\frac{\partial\phi_g(r,t)}{\partial t} = \sum_{g'=1}^{G}\Sigma_{sg'g}(r,t)\phi_g(r,t) + (1-\beta)\chi_g^P\sum_{g'=1}^{G}\nu_{g'}\Sigma_{fg'}(r,t)\phi_{g'}(r,t) + \sum_{i=1}^{I^{(D)}}\chi_{gi}^D\lambda_i C_i(r,t) + \nabla\cdot D_g(r,t)\nabla\phi_g(r,t) - \Sigma_{tg}(r,t)\phi_g(r,t) + S_{ext,g}(r,t) \qquad \text{Equation 2.22}$$

In these equations, *β* is the delayed neutron fraction, superscript *P* denotes prompt neutrons, *i* is the precursor group number, superscript *D* represents delayed neutrons, *C* represents delayed precursor concentration, and *λ* represents decay constant. The change in delayed neutron precursors is given by

$$\frac{\partial C_i(r,t)}{\partial t} = \beta_i\sum_{g=1}^{G}\nu_g\Sigma_{fg}(r,t)\,\phi_g(r,t) - \lambda_i C_i(r,t) \qquad \text{Equation 2.23}$$

If one wishes to initiate an eigenvalue-based transient, the fission term is divided by the eigenvalue, which forces the transient to start the system from a critical state (eigenvalue is unity). The problem is then solved at each time step as a fixed source problem. In NESTLE, a fixed source problem (FSP) backward difference method is used to ensure stability.

The time-dependent Boltzmann transport equation with explicit representation of delayed neutrons is given by Equation 1.4 where precursor rate of change is expressed by:

$$\frac{\partial}{\partial t}C_i(r,t) = \int_{4\pi}d\widehat{\Omega}'\int_0^\infty dE'\nu(r,E')\beta_i\Sigma_f(r,E',t)\Phi\left(\vec{r},E,\widehat{\Omega},t\right) - \lambda_i C_i(r,t) \qquad \text{Equation 2.24}$$

One example of this formulation is given by Goluoglu (13), which describes a quasi-static implementation for a time-dependent transport code called TDTORT. This is the formulation that is handled in TDTORT through the improved-quasi-static method, after some derivation and rearrangement (13).

Chapter 3 DERIVATIONS

Neutron kinetics methods focus upon the discretization of time in the time-dependent Boltzmann equation. Various methods also handle the discretization of space and angle differently, while neutron energy for this work's purposes is treated in a multi-group fashion. In order to meet the stated objective of this dissertation, which is the development of a robust, accurate, and computationally efficient transport-based space-time neutron kinetics method for deployment in a multi-physics framework capable of providing space-time thermo-mechanical-hydraulic feedback, a number of other neutron kinetics methods and discretizations of space, time, angle, and neutron energy are explored. In this chapter, we derive the two ends of the spatial neutron kinetics spectrum, which are factorization methods (quasi-static kinetics) and fully-coupled methods (direct kinetics). We then proceed with an example hybrid method (HOBD), as well as the developed hybrid method SIDK.

In the new generation of high performance computing, it is desired to develop new algorithms with multi-physics based predictive capabilities to improve upon legacy empirical models. These new predictive capabilities should be based on scientific first principles allowing for the simulation of failure scenarios. These scenarios should be based on actual failures, so to enable new insights at the design stage that could potentially mitigate these mechanisms in the future, as well as avoiding the related failures. An example of these types of failures are the fuel failures that occurred at the Edwing I. Hatch nuclear power plant during a routine control blade movement (10), a pellet-clad-interaction (PCI) event which could have been potentially identified a priori had the proposed multi-physics tools been available to evaluate the impact of blade movement and BWR channel bow upon the kW/ft generated at the axial height where the failures occurred. Clearly, the targeted application for these integrated multi-physics capabilities is nuclear fuel performance and safety, modeled in an integrated environment. This includes the software integration with focus upon the verification and validation of selected couplings as these physics are assembled.

The software integration herein described is between historically decoupled nuclear models of neutron transport and reactor kinetics with thermo-mechanical-chemical fuel performance models in the AMP framework. Two kinetics methods are implemented and tested within the AMP framework, which are the BRISC point kinetics package as well as a new implementation of semi-implicit direct kinetics (SIDK) that couples to the Denovo radiation transport package. The point kinetics derivations for BRISC are included in Chapter 4 as point kinetics is relatively straight forward and the derivation of the spatially independent equations is readily obtainable in any nuclear engineering textbook, such as (1) or (58).

Key physics involved in accurate prediction of reactor fuel element performance include neutron transport and thermal hydraulics. The thermal hydraulic feedback mechanism is primarily coupled to neutronics through cross sections, given that these are strong functions of temperature and density. Historically, this type of coupling has been routinely employed in nodal reactor core simulators, however, these generally capture thermal hydraulic conditions in coarse dimensions (typical node size may be a 6-inch axial section of a BWR bundle), thus giving a coarse treatment to individual fuel pins. In addition, these models need to be streamlined for quick execution, thus, they tend to be simplified models for thermal hydraulic calculations (a BWR simulation likely employs an approximate drift-flux model to model two-phase flow). The poor resolution on the primary coupling mechanisms can lead to unnecessary conservatisms that could be removed in order to improve fuel design and performance.

This dissertation seeks to quantify appropriate discretization strategies for the time-dependent Boltzmann transport equation that can be readily and non-intrusively implemented within a multi-physics framework. Space, time, and angle discretization strategies are explored with regards to several codes and methods employing direct, quasi-static, HOBD and the SIDK method with and without thermal feedback in one and two-dimensional geometries to determine the most appropriate strategies for massively parallel multi-physics implementations in Chapter 5. The non-intrusive implementation of the SIDK method, derived in this chapter, with large scale heterogeneous geometry neutron kinetics calculations employing an arbitrary number of neutron energy groups with variable order quadrature representation of angle is shown in Chapter 6. The coupling between the neutron transport and the thermal feedback is extremely important in this highly coupled problem, as it is primarily applicable to reactivity induced accidents (RIA) and loss of coolant accidents (LOCA), which is demonstrated in Chapter 5. An improvement in resolution and coupling is herein proposed by developing neutron transport models that are ideally suited for internal coupling with high fidelity within fuel pin thermal calculations in a multi-physics framework. Accordingly, good agreement with benchmarks and problems from the literature is shown.

3.1 NEUTRON KINETICS AND THE SIDK DERIVATION

A new approach to transport-based neutron kinetics has been developed within this dissertation to facilitate high performance computing through reduced storage requirements, speed of calculation, and ease of implementation in modern parallel, high fidelity steady-state radiation transport codes. The new kinetics methodology is a semi-implicit direct kinetics method, called the SIDK method (11). The SIDK method is herein presented in contrast to the two primary methods of neutron kinetics in use today, which are direct methods or factorization methods (7) as well as another hybrid method, the HOBD method (9). The sections which follow describe the derivation of the primary factorization method used, which is improved quasi-static kinetics (13). Then, the derivation of the direct method (7) is provided followed by the HOBD hybrid method. Finally, the SIDK methodology is derived and described.

3.1.1 QUASI-STATIC KINETICS

In the quasi-static kinetics implementation, the time-dependent angular flux is assumed to be the product of a weakly time-dependent angular flux multiplied by an amplitude function that is only a function of time:

$$\psi(r,E,\Omega,t)=\Psi(r,E,\Omega,t)T(t),$$ Equation 3.1

where ψ is the true time-dependent angular flux, Ψ is the weakly time-dependent angular flux, and T is the amplitude function which is only a function of time. When Equation 3.1 is substituted into the time-dependent Boltzmann equation, Equation 3.2 is produced.

$$\frac{1}{v}\left(\frac{\partial\Psi}{\partial t}T(t)+\frac{\partial T}{\partial t}\Psi(t)\right)+\ \widehat{\Omega}\cdot\nabla\Psi\left(\vec{r},E,\widehat{\Omega},t\right)+\ \Sigma_t\left(\vec{r},E,\widehat{\Omega},t\right)\Psi\left(\vec{r},E,\widehat{\Omega},t\right)-\int_{4\pi}d\widehat{\Omega}'\int_0^\infty dE'\Sigma_s\left(\vec{r},E'\rightarrow E,\widehat{\Omega}\rightarrow\widehat{\Omega}',t\right)\Psi\left(\vec{r},E',\widehat{\Omega}',t\right)-\chi_p(r,E)(1-\beta)\int_{4\pi}d\widehat{\Omega}'\int_0^\infty dE'\nu(r,E')\Sigma_f(r,E',t)\Psi\left(\vec{r},E,\widehat{\Omega},t\right)=\frac{1}{T(t)}\sum_{i=1}^{I}\chi_i^D\lambda_iC_i(r,t)+S\left(\vec{r},E,\widehat{\Omega},t\right) \qquad \text{Equation 3.2}$$

In this equation, υ represents velocity, Ω represents the normal vector in each direction for each angle, Σ_t represents a macroscopic total cross section, Σ_s represents a macroscopic scattering cross section, χ_p represents prompt fission yield, β represents total delayed neutron fraction, ν represents neutrons per fission, Σ_f represents a macroscopic fission cross section, λ represents a decay constant, *C* represents a precursor concentration, *i* represents a precursor group, and χ_i represents a precursor neutron yield. In order to solve for the amplitude function, the point kinetics equations are typically used. There are some choices on how to obtain the amplitude function and the associated weighting parameters; however, typically a normalization condition is applied using the adjoint transport equation such that:

$$\iiint\frac{\Psi^*(r,E,\Omega)\Psi(r,E,\Omega,t)}{v(E)}dVdEd\Omega=constant \qquad \text{Equation 3.3}$$

Where Ψ* is the adjoint angular flux. The steady-state adjoint equation is given by:

$$-\widehat{\Omega}\cdot\nabla\Psi^*\left(\vec{r},E,\widehat{\Omega}\right)+\ \Sigma_t\left(\vec{r},E,\widehat{\Omega}\right)\Psi^*\left(\vec{r},E,\widehat{\Omega}\right)-\int_{4\pi}d\widehat{\Omega}'\int_0^\infty dE'\Sigma_s\left(\vec{r},E\rightarrow E',\widehat{\Omega}\rightarrow\widehat{\Omega}'\right)\Psi\left(\vec{r},E',\widehat{\Omega}'\right)-$$

$$\chi_p(r,E')\int_{4\pi}d\widehat{\Omega}'\int_0^\infty dE'\nu(r,E)\Sigma_f(r,E)\Psi^*\left(\vec{r},E',\widehat{\Omega}'\right)=0 \qquad \text{Equation 3.4}$$

After some manipulation, the amplitude equation is given by the point kinetics equations, where the point kinetics parameters are determined by a series of integrations over the phase space which can be found in Goluoglu (13). The point kinetics equations are given by:

$$\frac{\partial c_i(t)}{dt}=\frac{\beta_i}{\Lambda}T(t)-\lambda_ic_i(t) \qquad \text{Equation 3.5}$$

$\frac{\partial T(t)}{dt} = \frac{\rho(t)-\beta}{\Lambda} T(t) + \sum_{i=1}^{I} \lambda_i c_i(t)$ Equation 3.6

In the first equation, c is the spatially *independent* precursor concentration, t is time, β is the delayed neutron fraction, T(t) is the power (or amplitude function), and λ is the decay constant. In the second equation, T(t) represents the rate of change of the power as a function of time (the amplitude function), where ρ is the reactivity insertion as a function of time, and Λ is the mean generation time. The key of the factorization method relies on the derivation of the spatially *dependent* precursors, or C_i(r,t). In the following derivation, the variable dependences are suppressed for simplicity. The spatially dependent precursors are given by:

$\frac{\partial C_i(r,t)}{dt} = \int\int \beta_i \nu \Sigma_f \cdot T(t)\Psi\left(\vec{r}, E, \widehat{\Omega}, t\right) dE' d\Omega' - \lambda_i C_i(r,t).$ Equation 3.7

If we define the product of $\nu\Sigma_f$ and Ψ integrated over angle and energy to be proportional to power, P(t), then we arrive at the following first-order ODE:

$\frac{\partial C_i}{dt} + \lambda_i C_i = \beta_i T(t) P(t),$ Equation 3.8

which can be solved analytically with the use of an integrating factor to yield:

$\frac{\partial}{dt}\left(e^{\lambda_i t} C_i\right) = e^{\lambda_i t} \beta_i T(t) P(t).$ Equation 3.9

Upon application of the chain rule, we have:

$C(t) = C_0 e^{-\lambda_i t} + \frac{\beta_i}{\lambda_i} T(t) P(t) (1 - e^{-\lambda_i t}).$ Equation 3.10

3.1.2 DIRECT KINETICS

Starting from the Boltzmann transport equation in operator notation,

$$\frac{1}{v}\frac{\partial\psi}{dt} + L\psi = MSD\psi + (1-\beta)MFD\psi + \sum_{i=1}^{I}\chi_i\,\lambda_i C_i\,, \qquad \text{Equation 3.11}$$

where L is the leakage operator, S is the scattering matrix, F is the fission matrix, M and D are the angular prolongation and restriction matrices, respectively, and I is the number of precursor groups. The time-dependent change in angular flux is:

$$\frac{\partial\psi}{dt} = v\{[-L + MSD + (1-\beta)MFD]\psi + \sum_{i=1}^{I}\chi_i\,\lambda_i C_i\}, \qquad \text{Equation 3.12}$$

and the time-dependent change in precursors is defined by:

$$\frac{\partial C_i}{dt} = \beta_i FD\psi - \lambda_i C_i \qquad \text{Equation 3.13}$$

In order to solve these equations in a fully-coupled manner, we define a matrix that contains both variables:

$$Z = \begin{bmatrix}\psi\\ C\end{bmatrix}, \qquad \text{Equation 3.14}$$

where we can now define A, B, E, and G respectively:

$$A = v[-L + MSD + (1-\beta)MFD] \qquad \text{Equation 3.15}$$

$$B = v\sum_{i=1}^{I}\chi_i\,\lambda_i, \qquad \text{Equation 3.16}$$

$$E = \beta_i FD, \qquad \text{Equation 3.17}$$

and,

$$G = -\lambda_i. \qquad \text{Equation 3.18}$$

This allows us to write the matrix in compact form:

$$\frac{\partial Z}{dt} = \begin{bmatrix}A & B\\ E & G\end{bmatrix} Z = \begin{bmatrix}A\psi & +BC\\ E\psi & +GC\end{bmatrix}. \qquad \text{Equation 3.19}$$

We then proceed with a finite differencing scheme for time using a fully implicit formulation, whereby the right hand side of Equation 3.19 is evaluated at the future time step "i+1".

$$\frac{Z^{i+1}-Z^{i}}{\Delta t} = \begin{bmatrix}A\psi^{i+1} & +BC^{i+1}\\ E\psi^{i+1} & +GC^{i+1}\end{bmatrix}. \qquad \text{Equation 3.20}$$

After rearrangement, we have:

$$\begin{bmatrix} \left(\frac{1}{\Delta t}I - A\right) & -B \\ -E & \left(\frac{1}{\Delta t}I - G\right) \end{bmatrix} \begin{bmatrix} \psi^{i+1} \\ C^{i+1} \end{bmatrix} = \begin{bmatrix} \frac{1}{\Delta t}\psi^{i} \\ \frac{1}{\Delta t}C^{i} \end{bmatrix}. \qquad \text{Equation 3.21}$$

Yielding the above coupled matrix system of equations that is solved for a direct kinetics solution.

3.1.3 Higher Order Backward Discretization Kinetics

As stated, the SIDK method is not the first hybrid of kinetics methods, where hybrid refers to a combination of the direct and factorization approaches, where the predominant factorization method in use is the quasi-static kinetics method. Three examples of hybrid methods are the Crank-Nicholson method as implemented in PARCS (8), the unified approach to kinetics solutions (7), and the higher order backward discretization (HOBD) that will be presented here from Ginestar et al (9). It should be noted that all of the hybrid methods previously mentioned are applied to the time-dependent neutron diffusion equation, while the SIDK method developed herein is applied to both the time-dependent neutron diffusion equation and the time-dependent Boltzmann equation. The higher order backward discretization method is chosen for derivation here as in the literature reviewed, it was the most similar method to the SIDK method developed in this dissertation, as well as the possible applications of the higher order method.

There are three potential applications offered by the higher order backward method. These three applications include a linear representation of the power over the time step (as opposed to the constant power at the beginning of the time step used in the SIDK method), higher order treatment of the temporal derivative of angular flux (as opposed to the first order backward difference common to most kinetics methods), as well as an adaptive time stepping strategy. These applications could potentially improve accuracy or run time of the SIDK method, at the potential cost of run time or accuracy, respectively, which is explored in Chapter 5.

Starting from the time-dependent multi-group neutron diffusion equation, in operator notation:

$$\frac{1}{v}\frac{\partial\phi}{\mathrm{dt}} + \mathrm{L}\phi = \mathrm{S}\phi + (1-\beta)\mathrm{F}\phi + \sum_{i=1}^{I}\chi_i\,\lambda_i C_i. \qquad \text{Equation 3.22}$$

Where ϕ is the scalar flux, which is used in the diffusion formulation as there is no angular prolongation or restriction operators applied to the scalar flux which is solved for in the diffusion implementation, the neutron group precursor derivative with respect to time is given by:

$$\frac{\partial C_i}{dt} = \beta_i \mathrm{F}\phi - \lambda_i C_i \qquad \text{Equation 3.23}$$

The only differences in the derivation given in Ginestar (9) and this derivation are that the scattering term has been included in the leakage operator and the Ginestar derivation is for two groups, specifically. As noted in the SIDK and quasi-static derivations, the time-dependent change in precursors can be solved analytically with a selection of a representation of the power over the time step *Δt*, here expressed as *h*. While the SIDK method developed uses a constant power over the time step, where the constant power is the power at the beginning of the time step, the higher order backward method presented here from Ginestar (9) uses a linear representation of the power over the time step, where the linear representation is given by:

$$\mathrm{C}_i(\mathrm{r}, \mathrm{t}+1) = \mathrm{C}_{i,\mathrm{t}}\mathrm{e}^{-\lambda_i \mathrm{h}} + \beta_\mathrm{i}\left(a_i \mathrm{F}^j \phi^j + b_i \mathrm{F}^{j+1}\phi^{j+1}\right), \qquad \text{Equation 3.24}$$

where *a* and *b* are coefficients given by:

$$a_i = \frac{(1+\lambda_i h)(1-\mathrm{e}^{-\lambda_i h})}{\lambda_i{}^2 h} - \frac{1}{\lambda_i}, b_i = \frac{\lambda_i h - 1 + \mathrm{e}^{-\lambda_i h}}{\lambda_i{}^2 h}. \qquad \text{Equation 3.25}$$

As mentioned, a variable time stepping strategy is presented in Ginestar (9) which is applied to the SIDK method with some modification, using the higher order discretization of the temporal flux derivative. Alternatively, the higher order representations of the power over the time step from either Ginestar (9) or PARCS (8) can be used with an arbitrary order of the temporal flux derivative, which is shown for the HOBD method in Chapter 5. Lastly, it is possible to use the higher order method to estimate the error of the lower order method, which allows the development of an adaptive time stepping strategy based on the error estimator provided by the higher order calculation. Examples of adaptive methods used in this work include the multi-step predictor-corrector approach used in MATLAB's implementation of a Runge-Kutta fourth and fifth order method (RK45) (23), which is discussed and compared to Pade approximant solutions for matrix exponential solutions in section 4.3 , as well as the NDF and TR-BDF2 methods (21) (30).

3.1.4 SEMI-IMPLICIT DIRECT KINETICS

In order to derive the developed SIDK method, we will represent some of the proceeding equations for flow and consistency highlighting the new method. Starting again from the time-dependent Boltzmann transport equation in operator notation, we have:

$$\frac{1}{v}\frac{\partial\psi}{\mathrm{dt}} + \mathrm{L}\psi = \mathrm{MSD}\psi + (1-\beta)\mathrm{MFD}\psi + \sum_{i=1}^{I}\chi_i\,\lambda_i C_i. \qquad \text{Equation 3.26}$$

If we take the direct kinetics matrix and assign every term implicitly (at time j+1) <u>with the exception</u> of the precursors (at time j), and apply a backward finite difference approximation to the derivative of the angular flux with respect to time, then we arrive at:

$$\frac{\psi^{j+1}-\psi^{j}}{v\Delta t} + L\psi^{j+1} = MSD\psi^{j+1} + (1-\beta)MFD\psi^{j+1} + \sum_{i=1}^{I}\chi_i\,\lambda_i C_i^{j}. \qquad \text{Equation 3.27}$$

Which is consistent with the angular flux formulation from direct kinetics (Equation 3.21) as shown below:

$$\begin{bmatrix}\left(\frac{1}{\Delta t}I - A\right) & 0 \\ -E & \left(\frac{1}{\Delta t}I - G\right)\end{bmatrix}\begin{bmatrix}\psi^{i+1} \\ C^{i+1}\end{bmatrix} + \begin{bmatrix}0 & -B \\ 0 & 0\end{bmatrix}\begin{bmatrix}\psi^{i} \\ C^{i}\end{bmatrix} = \begin{bmatrix}\frac{1}{\Delta t}\psi^{i} \\ \frac{1}{\Delta t}C^{i}\end{bmatrix}. \qquad \text{Equation 3.28}$$

Where it should be noted that representing the precursors explicitly allows us to solve the transport equation separately from the precursor equation. Finally, as many transport codes do not save the angular fluxes from one step to the next, the previous angular flux is approximated as the zeroth scalar moment:

$$\left[\frac{1}{v\Delta t} + L - MSD - (1-\beta)MFD\right]\psi^{j+1} = \sum_{i=1}^{I}\chi_i\,\lambda_i C_i^{j} + \frac{\emptyset^{j}}{4\pi v\Delta t}. \qquad \text{Equation 3.29}$$

Or alternatively, using the direct kinetics notation we have:

$$\left(\frac{1}{\Delta t}I - A\right)\psi^{i+1} = BC^{i} + \frac{\emptyset^{j}}{4\pi\Delta t} \quad . \qquad \text{Equation 3.30}$$

We now require a representation for the time-dependent precursors in order to solve for C^{i+1}. We will borrow this representation from quasi-static kinetics, without the use of weighting functions. This makes the semi-implicit direct kinetics method a hybrid in the sense that the same number of time steps as direct kinetics are used, but the system required to solve is much smaller. In contrast, a larger number of time steps than the quasi-static method typically employs are used; however, there are no adjoints or other weighting parameters that are problem-dependent that need to be solved. So, as the transport equation can now be solved independently of the precursor equations, the precursors at time t can now be solved analytically, following the quasi-static derivation without the introduction of the amplitude function. The spatial and group dependent precursor concentrations as a function of time simply become:

$$C_i(r,t) = C_{i,0}e^{-\lambda_i t} + \frac{1}{\lambda_i}\beta_i P(r,t)(1 - e^{-\lambda_i t}) \qquad \text{Equation 3.31}$$

There are two major advantages to this implementation and two drawbacks. The two advantages are: the angular fluxes from each time step do not need to be saved, which significantly reduces memory requirements. Also, the equations are now sequentially solvable, which further reduces memory requirements and run time. The first drawback to this implementation is that precursors are represented explicitly, which may require finer time steps than those typical of direct kinetics and could impact run time adversely. The potential benefits and disadvantages of this approach are explored in Chapter 5. In addition to this drawback, the time-dependent change in angular flux has been represented isotropically, which is not a good assumption in areas where there is a high degree of anisotropy. However, for typical reactor applications, the flux is largely isotropic, which has been demonstrated in other works (11) (59). Although anisotropic behavior cannot be fully represented, as reactors are largely isotropic in nature, this approximation is expected to be mitigated by memory savings, which are significant (14).

Chapter 4 POINT KINETICS

The simplest approximation of transient reactor response is to calculate an overall, or average, power change using reactivity coefficients. The process to obtain reactivity coefficients and other fundamental nuclear data is illustrated in Section 4.2 for various non-benchmark scenarios for which these data is needed and not provided. Using the point kinetics ODEs, it is possible to calculate an average reactor response. Two cases are examined for point kinetics without feedback, as well as an additional case for point kinetics with thermal feedback. These cases encompass verification via code to code comparisons, analytical problems, and benchmark comparisons. This work seeks to demonstrate the transient thermal evaluation capability that is present in AMP by interacting with point kinetics.

4.1 BURNER REACTOR INTEGRATED SAFETY CODE (BRISC)

The BRISC (Burner Reactor Integrated Safety Code) package has two primary components that run both in C++ and in FORTRAN. The first component of the BRISC package is RASCAL, which is a 2D diffusion code. RASCAL takes geometry input, as well as energy-collapsed and region-homogenized neutron cross section data, which is obtained from running a lattice physics code such as SCALE, in order to get cross sections such as $\nu\Sigma_f$, Σ_a, Σ_{tr}, etc. These cross sections are then used in conjunction with the geometry file to generate a cell-dependent fission source using RASCAL. The geometry file is compatible with either vacuum or symmetric boundary conditions. RASCAL also computes the critical eigenvalues (k_{eff}), as well as a fission source, as mentioned above.

The kinetics package then provides the user with power changes as a function of time. The inputs to this package include reactivity coefficients. Some of these reactivity coefficients (such as original fuel temperature) can be obtained from AMP quasi-static calculations. There are two solution methods employed by the kinetics package; the Crank-Nicholson approach and the Pade approximant. The Crank-Nicholson solution method uses two inputs. These are the number of sub-cycles and the alpha value, where the alpha value represents a range between forward Euler, backward Euler, and Crank-Nicholson. The default value of alpha=1 denotes an implicit method which utilizes a backward Euler method.

4.2 Development Of Nuclear Data

To begin the integration of the kinetics module from BRISC (KMB) with AMP, the SCALE code was used in order to generate relevant inputs. Typically, when developing a point or spatial kinetics code, computational benchmarks are selected, which include reactivity coefficients as well as beta and lambda values. Initial temperatures and thermal feedback models (if present) are also given. However, from an applications perspective, this data must be generated for real world problems. This section illustrates the process employed to obtain the basic nuclear data needed.

The main inputs needed for the kinetics calculations and subsequent comparisons were the reactivity coefficients and the delayed neutron fraction (β) values. Thus, a 2D NEWT calculation was performed to obtain k_{eff} and β values. The NEWT input can be found in APPENDIX A. From the base case (at the first time step conditions), k_{eff} was found to be 1.0370. So to obtain reactivity coefficients, it is necessary to perturb the fuel temperature and determine the reactivity response. This was done by raising and lowering the fuel temperature by ±300K while other temperatures were not changed. For this data generation, a pin cell case was run, consisting of a fuel rod, the helium gap, the zirconium-4 cladding, and heavy water. A square cell with sides equal to the pitch was used in conjunction with reflecting boundary conditions; which effectively assumes infinitely repeated pin cells, no neutron leakage and, thus, $k_{infinity}$ equal to k_{eff}. As this is a very simple case, a 4x4 and 8x8 mesh were employed, but this showed no effect on the results. The pin cell, mesh, and results obtained are shown below in Figure 3 through Figure 5, as well as in Table 1 and Table 2.

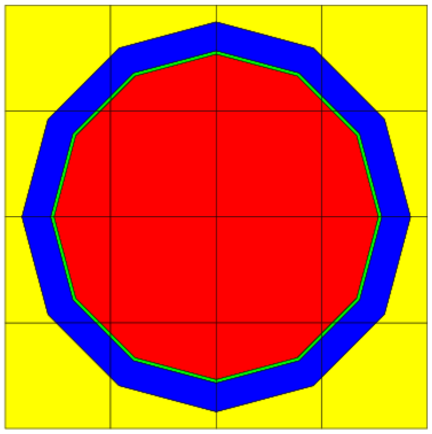

Figure 3 Geometry from pin cell case, 4*4 mesh.

Table 1: Delayed Neutron Fractions and Decay Constants

Delayed Group	Spectrum Constant (Beta)	Neutron Decay Lambda	Mean Generation Time (Λ)
	Fraction	(1/s)	s
1	2.40E-04	1.27E-02	
2	1.45E-03	3.17E-02	
3	1.34E-03	1.17E-01	
4	2.94E-03	3.15E-01	
5	1.07E-03	1.39E+00	
6	2.65E-04	3.85E+00	
Total	7.31E-03		1.00E-4

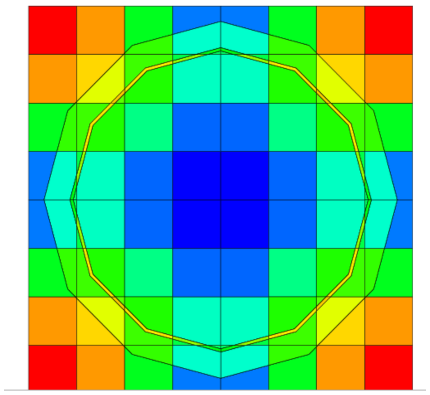

Figure 4 Group 1 fluxes (first 200 groups collapsed)

Table 2: Reactivity as a function of Fuel Temperature

k_{eff}	Reactivity	T_{fuel} (K)
1.061	0.0575	300
1.037	0.0357	607
1.021	0.0206	900

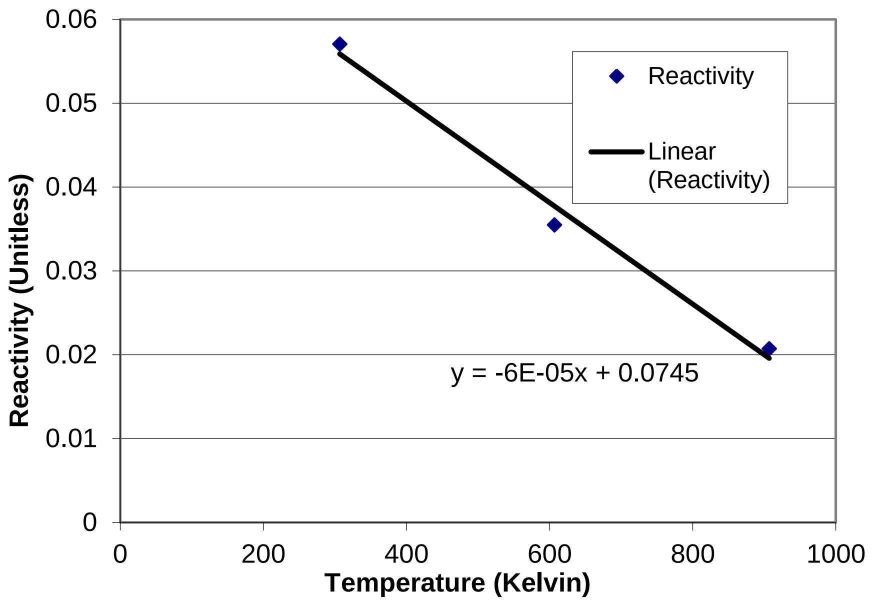

Figure 5 Reactivity as a function of fuel temperature

The Reactivity Coefficient (T_{fuel}) obtained from Figure 5 is $-6*10^{-5}$ (per degree Kelvin), which is small and negative similar to the typical value for a CANDU fuel reactivity coefficient (60). After the reactivity coefficient for the fuel temperature was determined, a few other reactivity coefficients were needed as inputs into KMB for the transient being run in KMB. This transient would alter several of the temperatures simultaneously, which is fairly realistic; if you have something like a loss of coolant accident (LOCA), the remaining coolant temperature will be elevated along with the structure, gap, and fuel temperature. The next temperature reactivity coefficient was found for the structure, which is the zirconium cladding. As expected, the reactivity coefficient for the zirconium cladding is smaller than that for the fuel temperature. This is shown below in Table 3. Next, the moderator temperature reactivity coefficient was determined. This was done in a similar manner to the structure, or clad, temperature reactivity coefficient. The moderator temperature appeared to have a negligible and unclear effect upon reactivity, as is seen Table 4 below, where both calculations estimated the reactivity coefficient at -2E-06 (per degree Kelvin).

Table 3: Reactivity as a function of Structure Temperature

k_{eff}	Reactivity	$T_{structure}$ (K)
1.03699	0.035673	520
1.03698	0.035664	525
1.03697	0.035656	530

Reactivity Coefficient (T_{clad}) =-2E-06 (per degree Kelvin)

Table 4: Reactivity as a function of Moderator Temperature

k_{eff}	Reactivity	T_{mod} (K)
1.03699	0.035673	508
1.03698	0.035664	513

Reactivity Coefficient ($T_{moderator}$) = -2E-06

4.3 Point Kinetics Without Feedback

The point-kinetics equations are a set of ordinary differential equations (ODEs) (58), where there is one equation for the power and one additional equation for each precursor group used. These equations were presented in section 3.1.1 as Equations 3.5 and 3.6. If thermal feedback is neglected, the point-kinetics equations predict an exponential jump in power followed by a slow exponential rise in power. The KMB code implemented within AMP is compared to an externally developed Runge-Kutta multistep predictor-corrector fourth- and fifth-order (RK45) code for comparison purposes as well as the NDF and TR-BDF2 time integration strategies. In addition, both codes are compared to a one-group formulation for the same problem, which can be solved analytically. The problem considered is an eight cent (where cent is percent of β_{eff}) step change positive reactivity insertion over a postulated ten second transient in a representative fuel pin from the Halden research reactor (61) during startup (low power). Figure 6 and Table 5 provide independent verification of the point-kinetics package KMB implemented within AMP. Both solutions were shown as convergent under mesh refinement with comparable run times of less than 1 min with a time step size of 0.01 s for the problem shown.

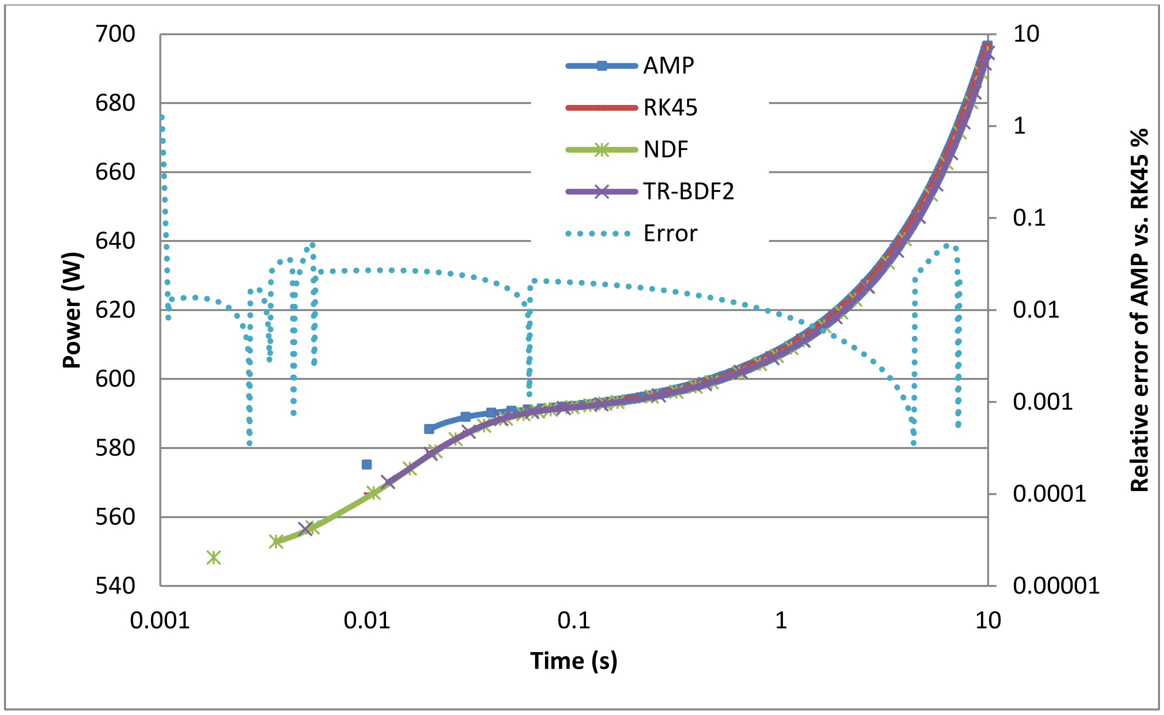

Figure 6 Point Kinetics Power Changes

Table 5: BRISC, RK45, Analytic, and AMP power changes at t=3 s

Quantity	Power (W)	Power Change (W)	Power Percent Error (from RK45)	Power Percent Error (from Analytic)
Six-group (BRISC)	631.28	88.28	-0.01%	1.72%
Six-group (NDF)	630.69	87.69	-0.1%	1.63 %
Six-group (TR-BDF2)	630.68	87.68	-0.1 %	1.63 %
Six-group (RK45)	631.21	88.21		1.71%
One-group (Analytic)	620.60	77.60	1.68%	
Six-group (AMP)	631.30	88.30	-0.01%	1.72%

4.4 Point Kinetics With Thermal Feedback

Using the point kinetics equations independently, it is possible to obtain power changes in a reactor in response to a given reactivity insertion. If these equations are decoupled from thermal feedback, the power first jumps rapidly, then grows approximately linearly. If a few seconds later, the power calculated is fed to a similarly decoupled thermal calculation, the mathematical prediction of thermal response can be several hundred degrees K for a modest reactivity insertion. In reality, the thermal response is not this drastic in reactors due to thermal feedback and the coupling of the thermal equations through this mechanism to the kinetics equations. Typical methods of handling this coupling include adding an ODE to the point kinetics equations or to calculate thermal response on a larger scale, using thermal-hydraulic response (62).

Including thermal feedback adds one additional ODE to the system, which is shown below.

$$\frac{dT(t)}{dt} = \frac{1}{c_p}[\varepsilon P(t) - \frac{AHD^{0.75}[T(t)-T_c]^{1.25}}{T(t)^{0.25}}]$$

Equation 4.1 (63)

In this equation, T is temperature, c_p is specific heat capacity, ε is the fraction of fission energy deposited as heat, H is height, D is diameter, A is a material-dependent constant, T_c is coolant temperature, and t is time (where the geometry considered is a cylinder and the heat transfer considered is natural convection.). For comparison purposes, the point-kinetics parameters developed by Dodds (63) were used with the point-kinetics-with-feedback model. The relevant parameters for the problem are presented below in Table 6 and Table 7:

Table 6 Point Kinetics with Feedback Problem Parameters

Parameter	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
β (Fraction)	2.2E-4	1.42E-3	1.27E-3	2.57E-3	7.5E-4	2.7E-4
λ (s)	1.24E-2	3.05E-2	1.11E-1	3.01E-1	1.14	3.01

Table 7 Point Kinetics with Feedback System Parameters

Parameter	Value	Units
P(t=0)	.01	W
ρ(t=0)	4.3	Cents
Λ	5e-5	s
D	0.2	m
H	0.23	m
A	17.52	constant
C_p	1.012e4	J/kg-C
ε	1.0	unitless
T_c and T(t=0)	20	C
$\Delta\rho$(t)	-0.306	Cent/K

The problem considered was a step change in reactivity using six neutron precursor groups and thermal feedback through a reactivity coefficient expressed in cents per degrees Celsius. The problem was run for 1000 minutes, simulating an extended transient longer than 10 hours. The code developed for this work used a Runge-Kutta multistep predictor corrector method using MATLAB, as well as the NDF and TR-BDF2 time integrators. This code is compared to SKINATH, developed by Dodds (63).

This method is similar to the methodology employed in AMP. However, the temperature distribution is calculated by a 3D transient thermal solver within AMP using the IDA (64) time integrator. This integrator drives both the thermal and power calculations in AMP.

The power and temperatures (system averaged) as a function of time from the reference SKINATH and the RK45 code developed are shown in Figure 7 and Figure 8. The RK45 time integrator uses a higher-order method relative to the first-order method (LSODE) used by SKINATH, which leads to a peak power predicted by the RK45 method that is lower than that predicted by SKINATH, as well as to a lag in time throughout the transient's peaks and valleys. However, in fact, it is observed that both of the variable higher order methods (NDF and TR-BDF2) also exhibit consistent behavior with the RK45 method. This agreement and the trends are consistent with the agreement demonstrated in Dodds (63) against an RK4 method. Similar trends and results are observed in the temperature plot presented in Figure 8.

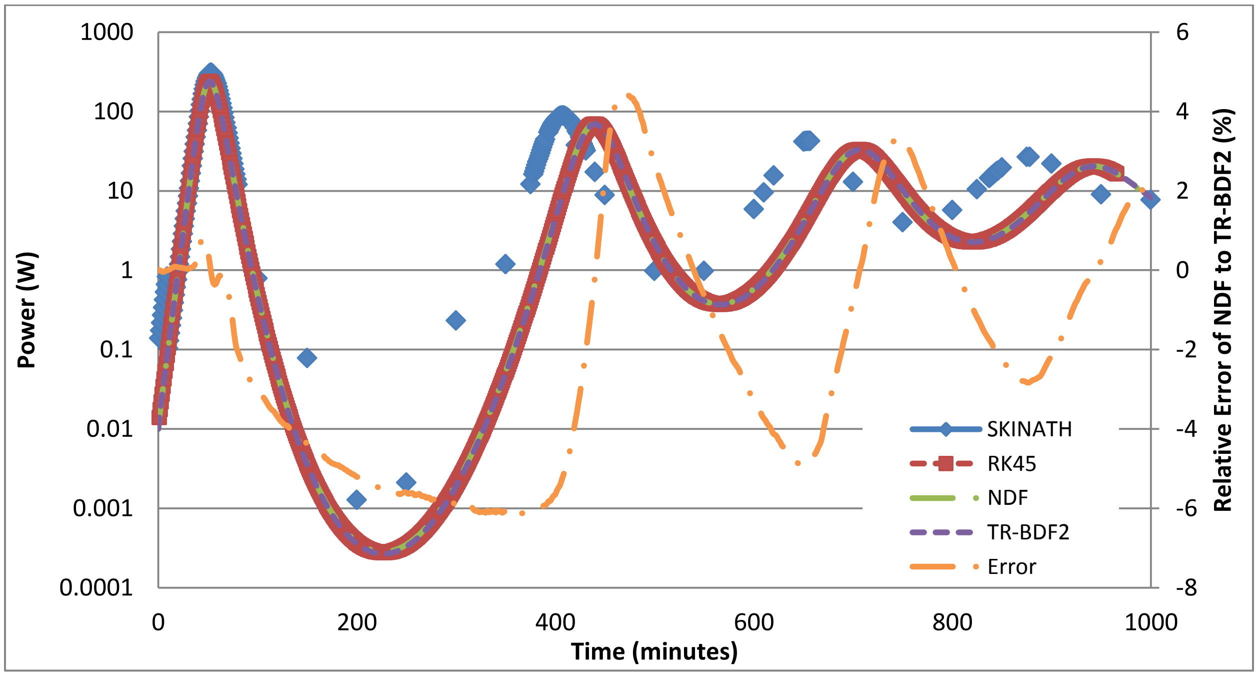

Figure 7 RK45 Point Kinetics with Thermal Feedback vs. SKINATH

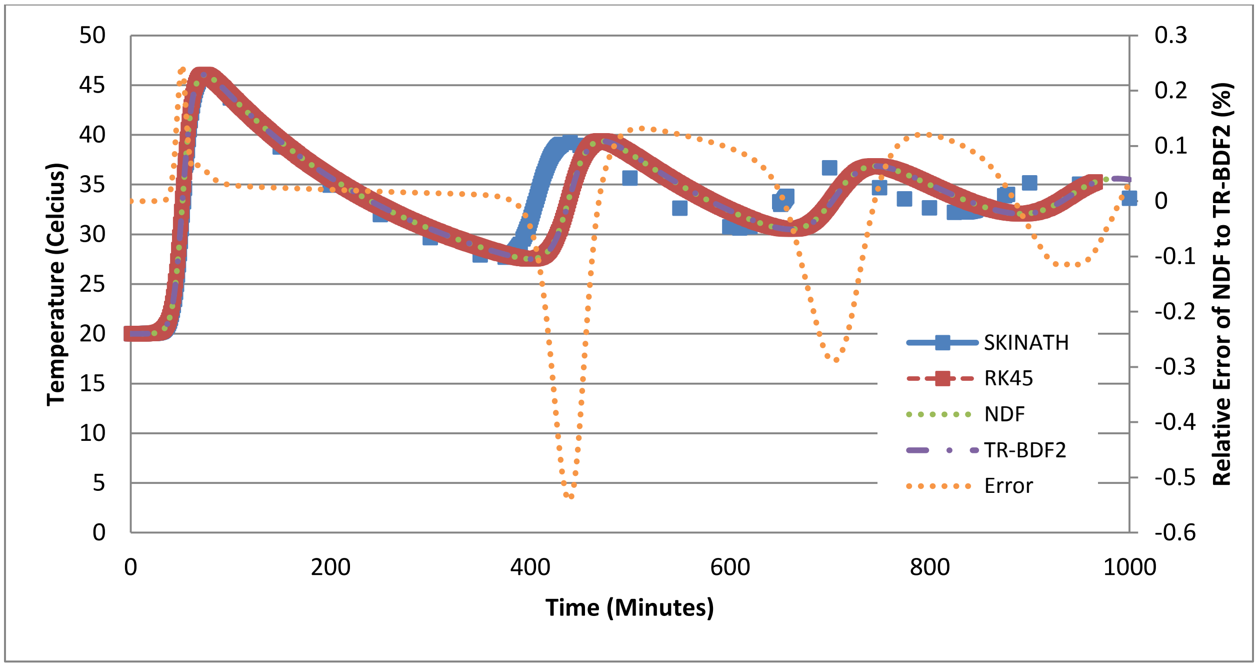

Figure 8 RK45 Temperature vs time vs. SKINATH

Chapter 5 TIME-DEPENDENT NEUTRON DIFFUSION

This chapter serves to explore the spatial and temporal discretization of the time-dependent Boltzmann neutron transport equation with and without thermal feedback in fast and thermal reactors on a core-wide scale. As such, for all problems considered in this chapter, independent two-group time-dependent neutron diffusion codes were written for comparison to two-group benchmarks that are primarily diffusion based, with the exception of the first benchmark considered. The first benchmark considered is a one-dimensional fast reactor which was computed, by the benchmark (12), utilizing quasi-static one-dimensional neutron transport. We will see the impact of using a neutron transport solution, where angular discretization is treated, by revisiting the same benchmark problem in Chapter 6 with the use of the Denovo radiation transport solver. The next two benchmark problems considered are reactivity insertions in two-dimensional thermal spectrum reactors, where the first problem has no thermal feedback and the second problem has thermal feedback. The problem without thermal feedback, the TWIGL problem, is used to consider a number of time discretization strategies including comparisons to neutron transport solutions, while the problem with thermal feedback, the LRA problem, is used to quantify errors introduced primarily by frequency of cross-section update, which is where the SIDK method is shown to be the strongest method, for medium error tolerances and spatial resolutions. The LRA problem represents the primary target problem of the developed SIDK method, which would be in a multi-physics framework where heterogeneous geometry including high fidelity neutron transport and thermal feedback calculations could be included.

As has been demonstrated in the literature (6), quasi-static kinetics is the fastest of the spatial kinetics methods when the flux shape changes slowly. However, if the flux shape is changing rapidly, such as during a prompt supercritical transient like a control rod ejection, quasi-static kinetics can in fact be slower than direct kinetics, due to the calculation of the collapsed point kinetics parameters that are not adding any performance (6). The SIDK method developed in this work does not share this approximation of slow shape change, as it uses the same fine time step that is typical of direct kinetics, which for medium error tolerances, can make the SIDK method faster than direct kinetics. For fine error tolerances, higher order methods are recommended. This is demonstrated independently by a 1D neutron diffusion program written by the author. This one dimensional diffusion program was developed to study the performance increase of the method developed versus direct kinetics for the 16-A1 benchmark. In addition, this section serves to illustrate the spatial convergence behavior of the SIDK method versus direct kinetics, while section 5.2 will illustrate the temporal convergence for various methods including higher order methods, and section 5.3 will illustrate the combined effects of spatial and temporal convergence in the presence of thermal feedback, which introduces temporal errors that are not due to the representation of the temporal derivative of the flux.

5.1 ONE-DIMENSIONAL FAST REACTOR BENCHMARK WITHOUT FEEDBACK (ANL-16-A1)

The first benchmark problem chosen is a one-dimensional fast reactor benchmark. This is an excellent problem to examine the impact of transport versus diffusion solutions, as the benchmark is one of the few transport based spatial kinetics benchmarks available in the literature. In addition, the reactor considered is a fast spectrum reactor, which have very different neutron mean generation times than thermal reactors (typical thermal reactor mean generation time is 1E-4s, while typical fast reactor mean generation time is 1E-7 s). The benchmark that was selected for this effort was ANL Benchmark problem 16-A1, which is a 1D spatial kinetics benchmark for a fast reactor (13) (12). In this section, a presentation of the problem description will be followed by a results subsection and an analysis subsection.

5.1.1 16-A1 PROBLEM DESCRIPTION

The problem features seven material regions and two energy groups, for which the cross sections are provided in the benchmark. The parameters for the problem are shown below in Table 8 through Table 11.

Table 8: Cross Sections for Benchmark Problem 16-A1

Zone	Group	Sigma F	Sigma T	Sigma g->g	Sigma g->g’
1 and 7	1	8.3441e-4	2.411e-1	2.336e-1	3.598e-3
	2	3.2776e-4	4.172e-1	4.07e-1	0
2,4, and 6	1	7.4518e-3	1.849e-1	1.777e-1	2.085e-3
	2	1.1061e-2	3.668e-1	3.537e-1	0
3 and 5	1	0	9.432e-2	8.571e-2	1.717e-3
	2	0	1.876e-1	1.713e-1	0

Table 9: Delayed Neutron Parameters for Benchmark Problem 16-A1

Delayed Neutron Group	Delayed Neutron Fraction, Beta	Decay Constant, lambda
1	8.1e-5	.0129
2	6.87e-4	.0311
3	6.12e-4	.134
4	1.138e-3	.331
5	5.12e-4	1.26
6	1.7e-4	3.21

Table 10: Point-Kinetics Parameters for Problem 16-A1

Reactivity	Total Effective Delayed Neutron Fraction	Mean Generation Time	Velocity Group 1	Velocity Group 2
1.110e-3	3.2e-3	3.655e-7 s	5.402e8 cm/s	9.191e7 cm/s

Table 11: Mesh Intervals for Problem 16-A1

Zone	Number of Intervals	Width
1	20	40 cm
2	24	47.374 cm
3	5	9 cm
4	16	34 cm
5	5	9 cm
6	24	47.374 cm
7	20	40 cm

This section serves to quantify the accuracy of the spatial discretization strategies implemented by the SIDK approach as well as to demonstrate the computational advantage of the method developed over using direct kinetics which is particularly large for this six neutron group precursor transient, so that the SIDK may serve as an intermediate option between direct kinetics and quasi-static kinetics methods.

5.1.2 16-A1 RESULTS

The eigenvalue was within 600 percent-mille (pcm) of the 16-A1 benchmark and was shown to be convergent under mesh refinement. An identical mesh to that presented in the 16-A1 benchmark yielded an eigenvalue of .991826, which is within 1000 pcm of the benchmark. This difference is primarily attributable to the fact that the 16-A1 benchmark is a transport benchmark. We will see in Chapter 6 this error removed when utilized the Denovo code. The eigenvalues under spatial mesh refinement are shown in Table 12 while a comparison of the steady-state fluxes and perturbed group 1 and 2 fluxes are shown below in Figure 9 and Figure 10.

Table 12 Eigenvalues of Diffusion program vs. 16-A1 Benchmark

Mesh	Eigenvalue	Error from Benchmark (pcm)	Error from fine mesh (pcm)
114	0.991826	837.2	184.9
228	0.992923	727.5	75.2
456	0.993431	676.7	24.4
912	0.993675	652.3	0

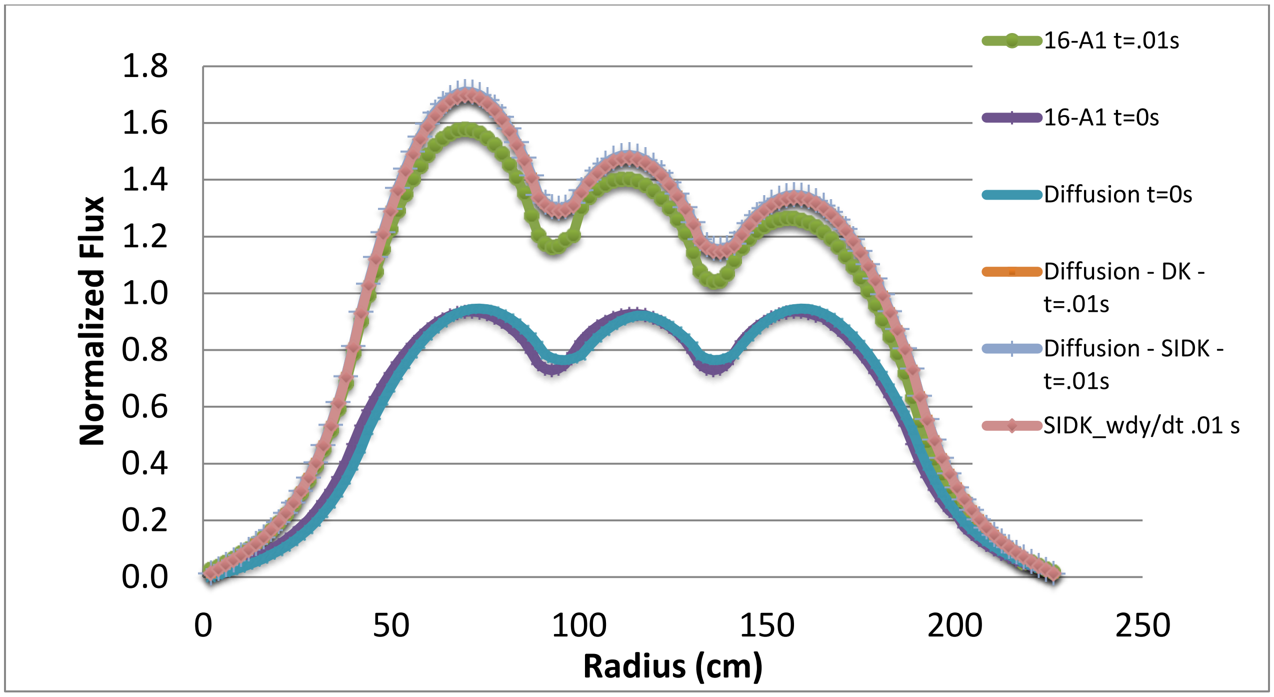

Figure 9 Diffusion - Group 1 Steady-State and Perturbed Fluxes

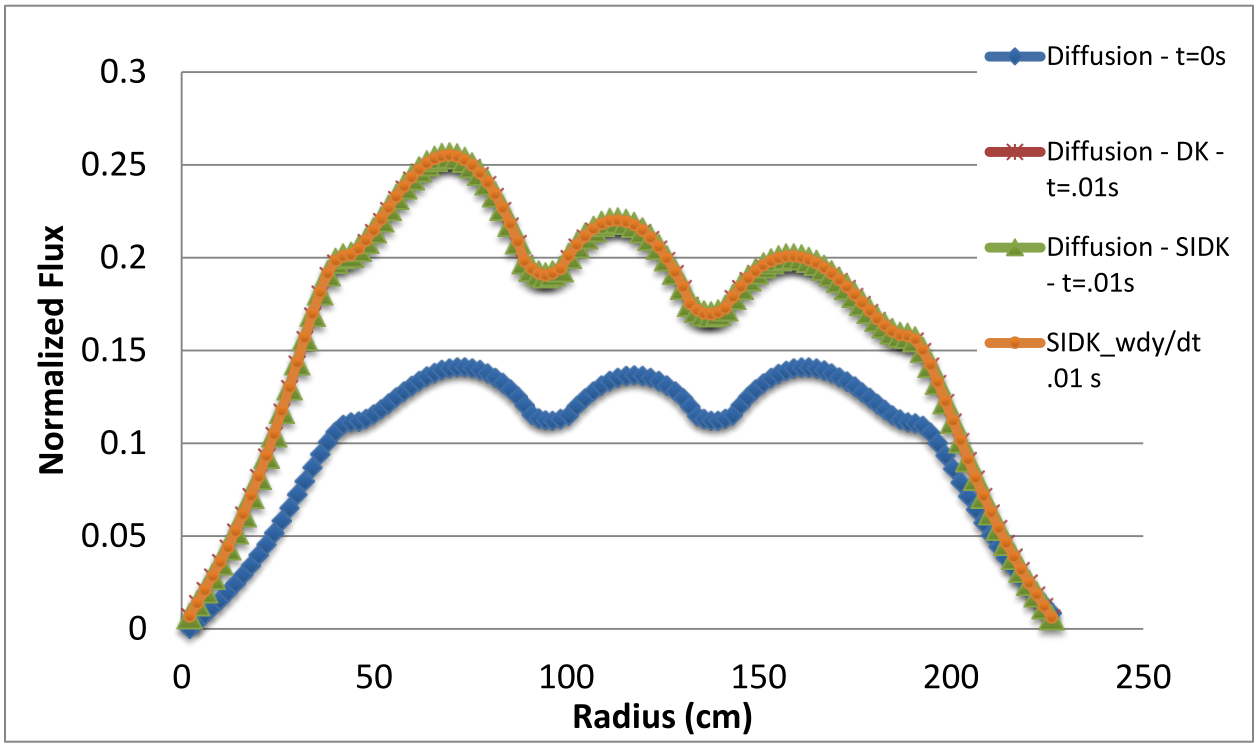

Figure 10 Diffusion - Group 2 Steady-State and Perturbed Fluxes

As it can be observed, all solutions employed in the SIDK diffusion code produce reasonable results with respect to the benchmark. It is also observed that the SIDK method is extremely close to the direct kinetics method, with a power rise within 0.2 % of the direct method. This difference is almost entirely attributable to the exponential treatment of precursor change in addition to utilizing the beginning of time step powers to compute the precursor change.

However, the semi-implicit direct kinetics method ran an order of magnitude faster than the direct method. This difference may be somewhat lessened if iterative instead of direct solvers were employed, which will be illustrated in section 5.3 ; however, it is still substantial. A speed comparison is shown below in Table 13, in which the speed-up is largely due to the fact that the problem is a two-group diffusion problem; hence, for the SIDK method, the matrix to be inverted is a matrix of rank=2*nodes. In contrast, for the fully implicit direct kinetics, there are six spatial precursors at every spatial point, causing the matrix to be inverted to be of rank=8*nodes, so having a 4 times larger rank matrix requires substantially greater computational effort to invert. It is also shown that this performance increase increases with spatial refinement, or as the problem size grows. It can also be observed in Figure 9 and Figure 10 that for this problem, the inclusion of the time-dependent change in flux (scalar flux for the diffusion case) made very little impact (less than 0.7 %) on the computed power magnitudes. This is due to the fact that this case is a fast reactor which has a very high velocity, which makes the temporal derivate of angular flux a less important term.

5.1.3 16-A1 ANALYSIS

Table 13 illustrates that using the same time step, spatial refinement causes a largely constant error between the SIDK method and direct kinetics, and that the inclusion of the time-dependent change in flux reduces this error. It is observed at finer spatial mesh sizes that this error is reduced, again for both cases. The fine-mesh finite-difference diffusion code implementation is shown to have quadratic convergence in space, which is the expected convergence of a finite-difference implementation. The following series of figures will compare the error of the SIDK method with and without the time-dependent change in angular flux to the fine spatial mesh direct kinetics solution, the 16-A1 benchmark point kinetics solution, as well as the 16-A1 benchmark spatially integrated solution, using the same time step size of 0.01 s.

Table 13: Speed and Accuracy for 16-A1 Comparison at t=0.01s at various mesh refinement levels

Mesh	Direct Kinetics (Reference) Normalized Power	Semi-Implicit Direct-Kinetics w/ dϕ/dt Normalized Power	Power Error (from Direct Kinetics in %)	Semi-Implicit Direct-Kinetics without dϕ/dt Normalized Power	Power Error (from Direct Kinetics in %)	Speed-up (SIDK to Direct)
114	1.603749	1.600722	0.1887	1.612184	0.5260	15.7
228	1.580238	1.577371	0.1814	1.588226	0.5055	25.8
456	1.568914	1.566122	0.1780	1.576690	0.4956	36.4
912	1.563355	1.560600	0.1762	1.571028	0.4908	36.6

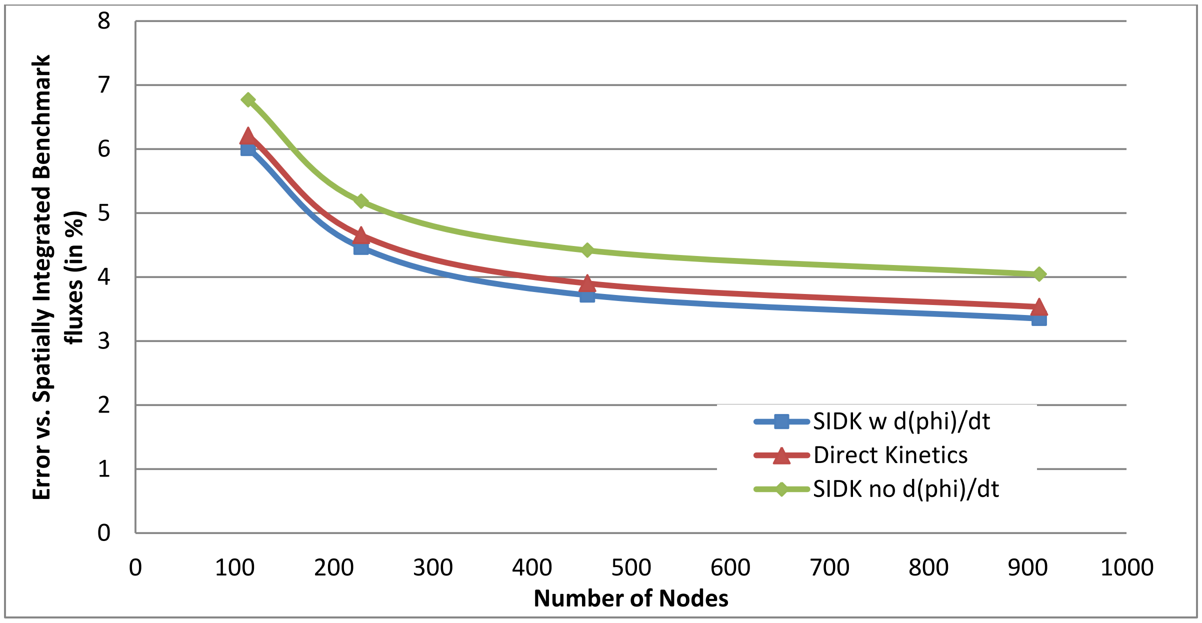

Figure 11 Errors of SIDK with and without dφ/dt and direct kinetics (diffusion implementation) versus spatially integrated benchmark powers at t=0.01s

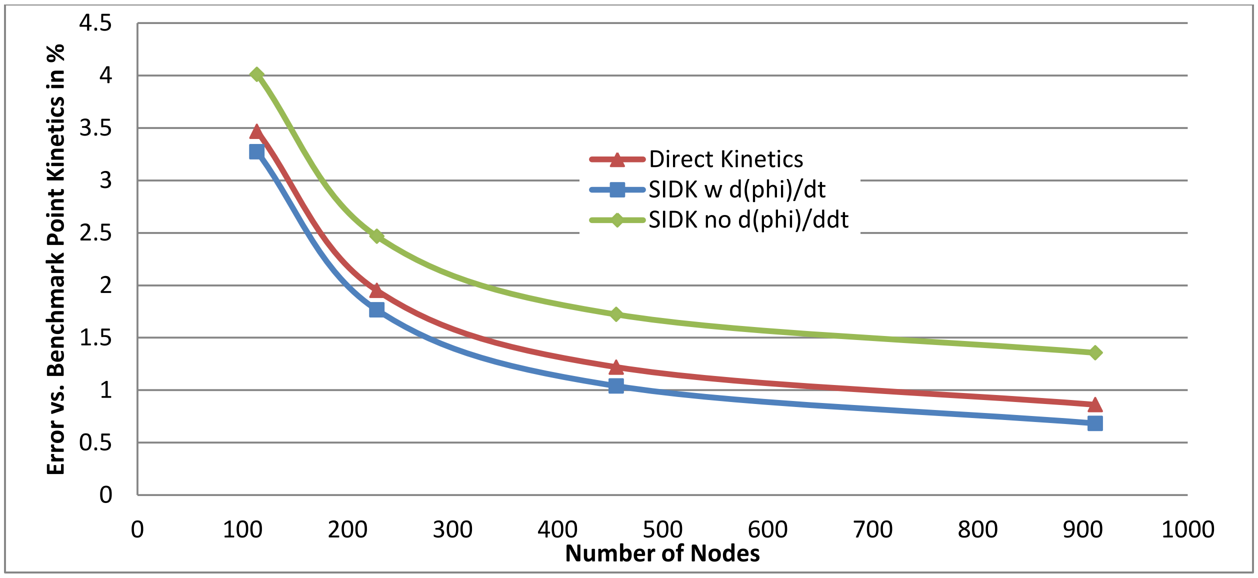

Figure 12 Errors of SIDK with and without dφ/dt and direct kinetics (diffusion implementation) versus point kinetics powers at t=0.01s

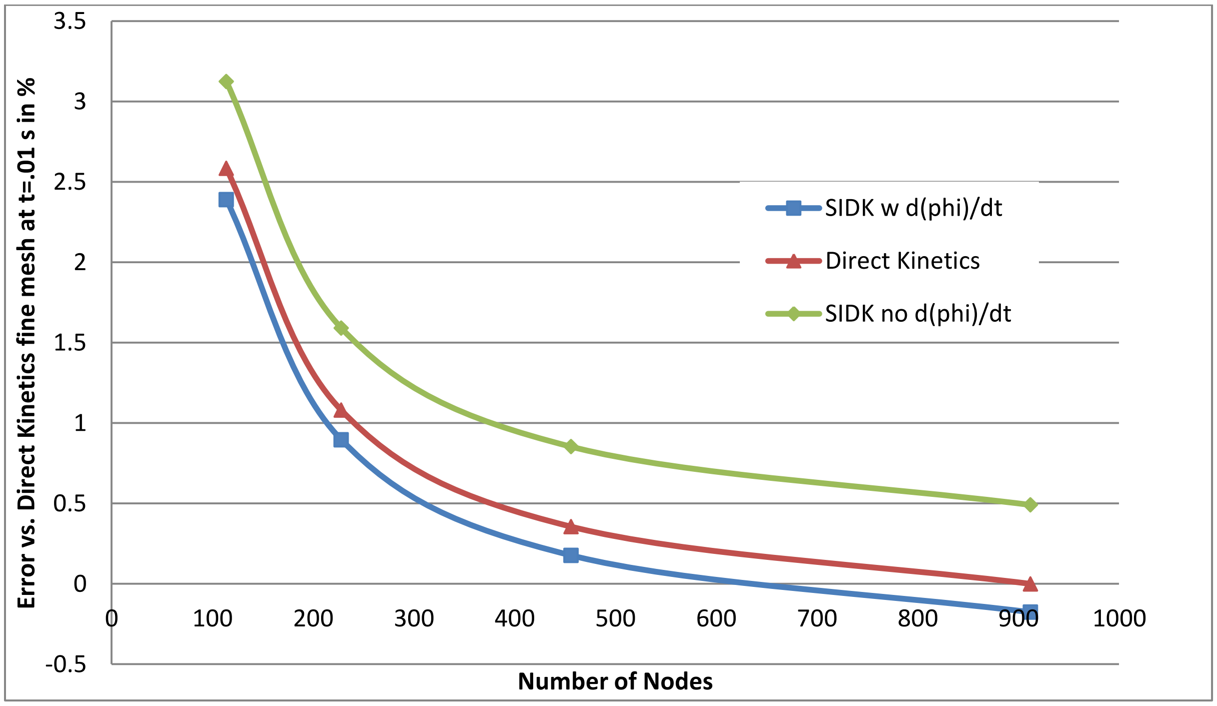

Figure 13 Errors of SIDK with and without dφ/dt and direct kinetics (diffusion implementation) versus fine mesh direct kinetics powers at t=0.01s

5.2 Two-dimensional TWIGL Seed-Blanket Reactor Problems without Thermal Feedback

The TWIGL problem (6) (13) was selected as a benchmark to demonstrate the performance of the method on problems without thermal feedback compared to a number of time integration strategies for two-dimensional geometries for thermal spectrum reactors. In this section, a problem description subsection will be followed by results and analysis subsections, which is concluded with an examination of higher order methods applied to the same problem.

5.2.1 TWIGL Problem Description

The compositions, cross sections, and delayed neutron data are as follows in Figure 14, Table 14, and Table 15, while the remaining specifications are shown below. Each block in Figure 14 is 8 cm by 8 cm.

$\nu = 2.43$
$v_1 = 1*10^7$ cm/s
$v_2 = 2*10^5$ cm/s
Dimensions = 80 X 80 cm

Perturbations

Step
$\Delta\Sigma a_2 = -.0035$ t=0 s
Ramp

$\Sigma a_2(t) / \Sigma a_2(t) =$	$1 - 0.11667*t$	$t<=0.2$ s
	0.97666	$t>0.2$ s

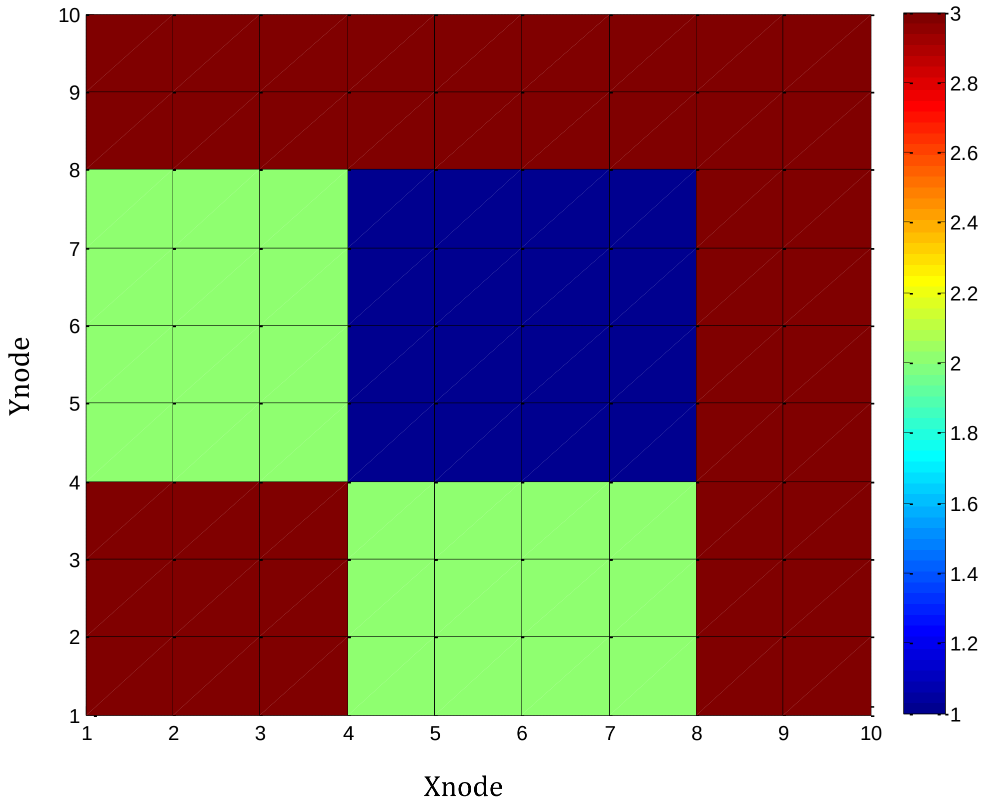

Figure 14 TWIGL Benchmark Compositions (Colors Denote 1, 2, and 3)

Table 14 TWIGL Composition Parameters

Composition	Group i	D_i (cm)	Σa_i (cm^{-1})	$\upsilon\Sigma f_i$ (cm^{-1})	$\Sigma_{1\rightarrow2}$ (cm^{-1})
1	1	1.4	0.01	0.007	0.01
	2	0.4	0.15	0.2	
2	1	1.4	0.01	0.007	0.01
	2	0.4	0.15	0.2	
3	1	1.3	0.008	0.003	0.01
	2	0.5	0.05	0.06	

Table 15 Delayed Neutron Data

Group	β_i	λ_i (s^{-1})
1	0.0075	0.08

5.2.2 TWIGL Results and Analysis

The eigenvalue computed with the developed two-dimensional two-group diffusion code (designed to test the SIDK method) was within 65 pcm of that reported in the benchmark. Spatial mesh refinement also caused greater agreement in transient results, demonstrated further for the LRA problem in section 5.3 . Figure 15 depicts the normalized power distribution in the reactor at time t=0 s. As can be observed in Figure 16 and Figure 17, the powers computed with the SIDK approach were extremely consistent with those computed in the same code using direct methods, while the results from TDTort and CONQUEST basically illustrate different methodologies employed, where TDTort is a 3D time-dependent transport code and CONQUEST is a 3D time-dependent nodal diffusion code, both using quasi-static kinetics. This specific comparison shows a two-fold advantage for the SIDK methodology in terms of computational speed relative to direct kinetics. The reason the SIDK method speed-up is not as substantial as that observed in the 16-A1 problem is due to the fact that this problem uses only one delayed neutron group, which reduces the additional size of the matrix utilized in direct kinetics.

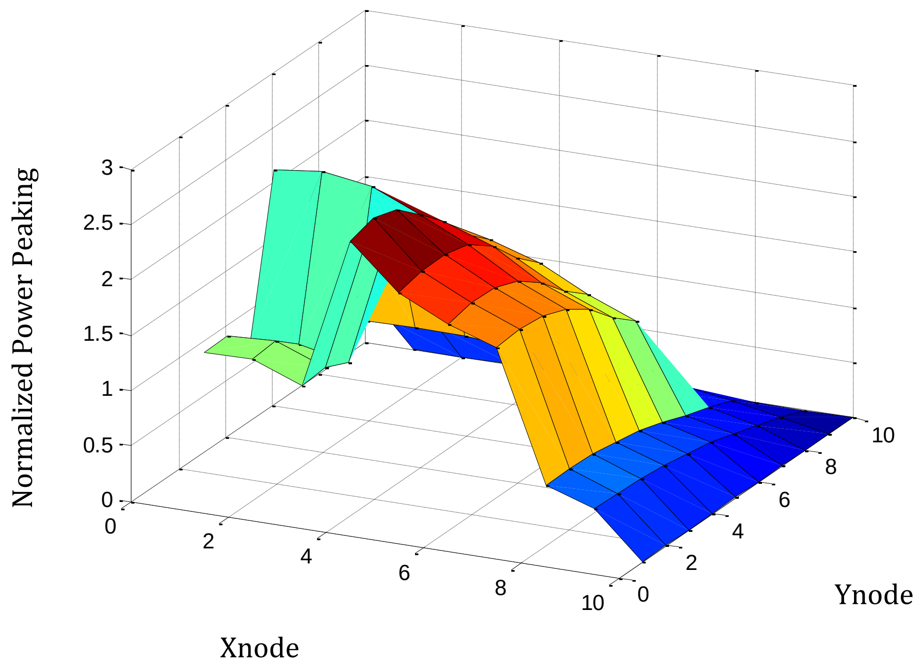

Figure 15 Normalized Power Distribution (t=0s)

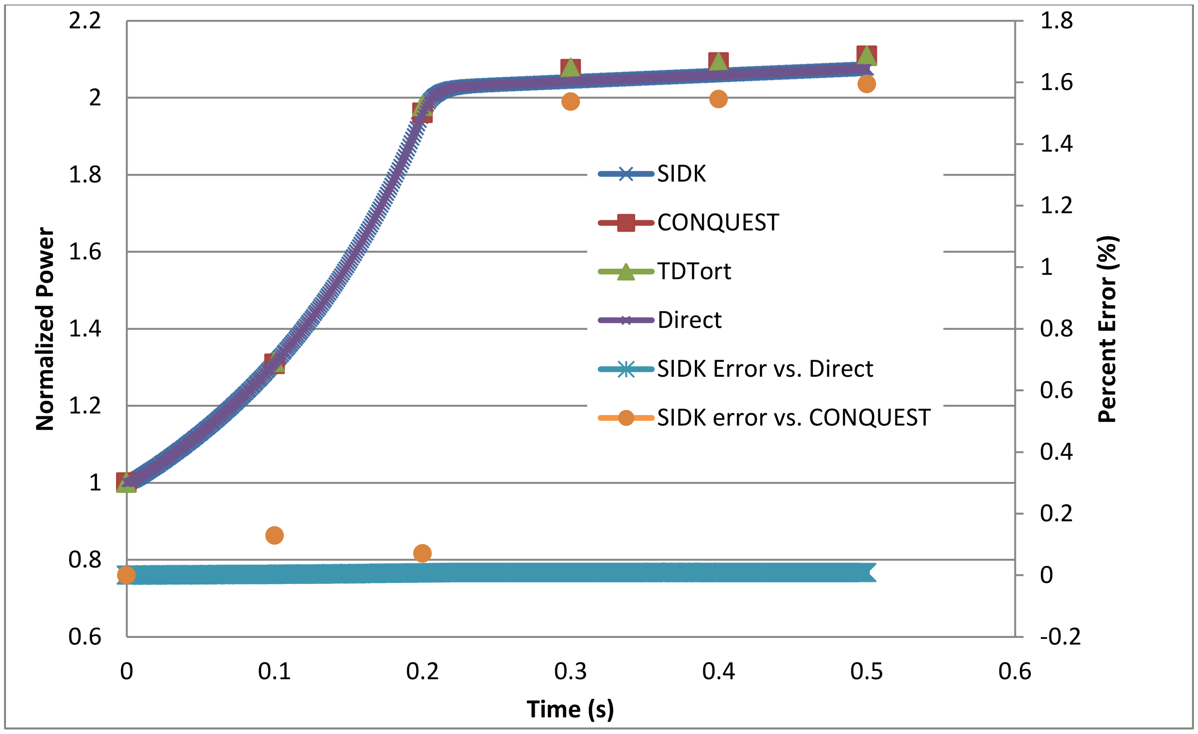

Figure 16 Ramp Results

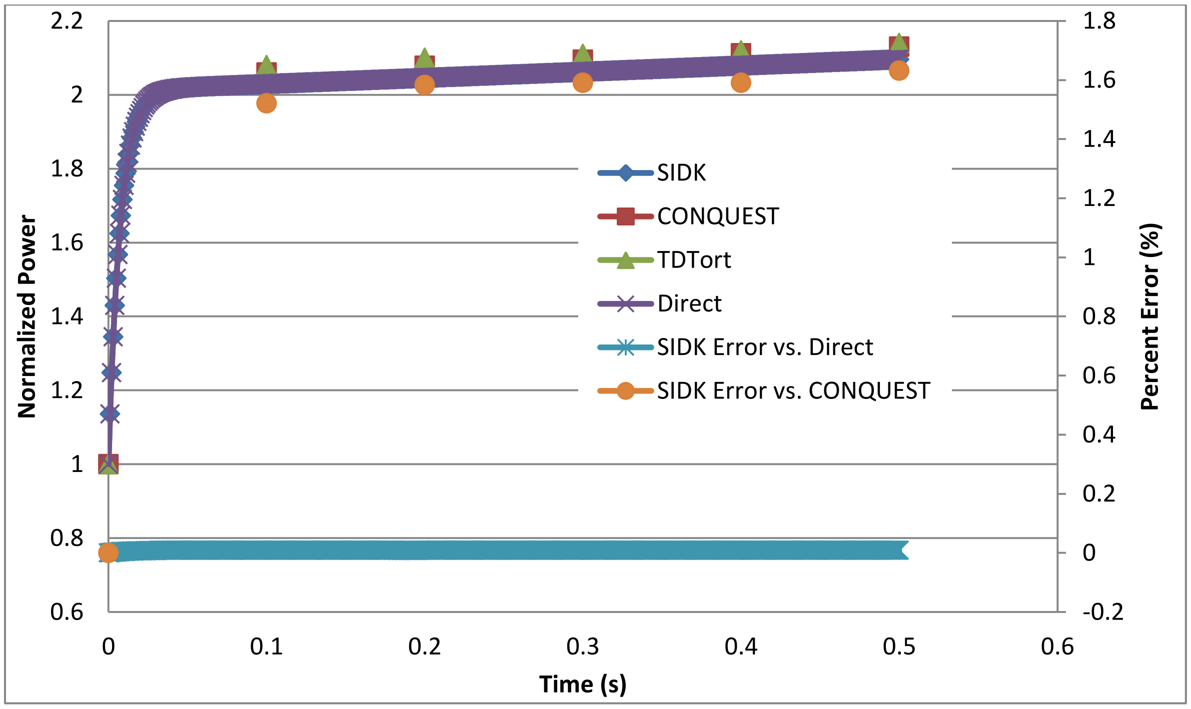

Figure 17 Step Results

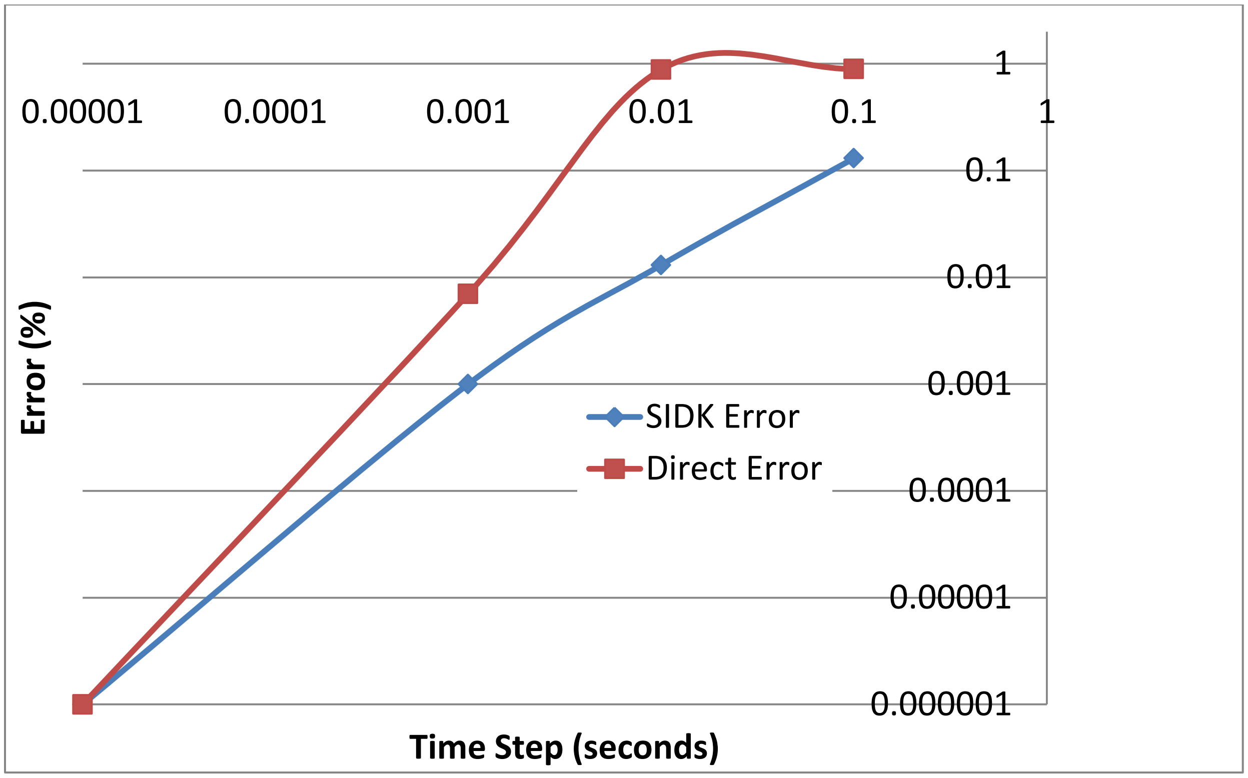

Figure 18 TWIGL step problem errors at t=0.5 s versus fine time step (1E-5s) direct kinetics

It is observed from Figure 18 that the SIDK method converges under temporal refinement to the fine mesh direct kinetics solution linearly and faster than the direct method, implying that the analytic representation of the precursor concentrations is in fact out-performing the first order finite difference approximation in direct kinetics, even though the direct kinetics representation is fully implicit. A speed comparison is shown below in Table 16.

Table 16: Speed and Accuracy for TWIGL Comparison

Method	Direct Kinetics (Reference, fully implicit)	Semi-Implicit Direct-Kinetics
Normalized Power	1.990	1.986
Error	N/A	0.2 %
Speed (s)	2 s	1 s
Speed-up (Relative to Reference, in x faster)	N/A	2 x faster

5.2.3 Higher-Order Representations

As the TWIGL problem is a well known, relatively simple problem, a number of perturbations were done on the problem in order to determine the advantages and disadvantages offered by the SIDK (11) as well as the HOBD (9) methods. In addition to these hybrid kinetics methods, different time integration strategies were also tested for the fully implicit direct solutions, including trapezoidal rule – backward difference formulation, second order (TR-BDF2) (23) (30) as well as a variable order solver based on the numerical differentiation formulas (NDFs) (21). Also, the data presented for a BDF6 method in (59) is included for comparison purposes. TR-BDF2 is a multi-stage, multi-step method which is an implicit Runge-Kutta method. The TR-BDF2 method uses two-stages which consist of a trapezoidal rule stage followed by a second-order BDF stage. Both the TR-BDF2 and the NDF methods are adaptive methods utilizing variable time stepping strategies with error estimation. The developed SIDK method is very similar to the HOBD method, with the main difference being that the HOBD method uses a semi-implicit representation of the power over the time step which is a weighted average of the initial and final powers, where the SIDK method uses an explicit representation of the power over the time step.

Both first and second order SIDK methods, using the multi-step approach to discretize the time-derivative of the flux, were used where the HOBD method only utilized a second-order formulation. The fully-implicit direct, SIDK methods and HOBD methods all utilized constant time steps. Figure 19 and Figure 20 are convergence and efficiency plots, respectively. The convergence plot is present in order to determine the actual order of convergence of each method. In Figure 19, it is demonstrated that for the fully-implicit direct kinetics, the first and second order SIDK, the second order HOBD, as well as the NDF solvers are all first order solution methods.

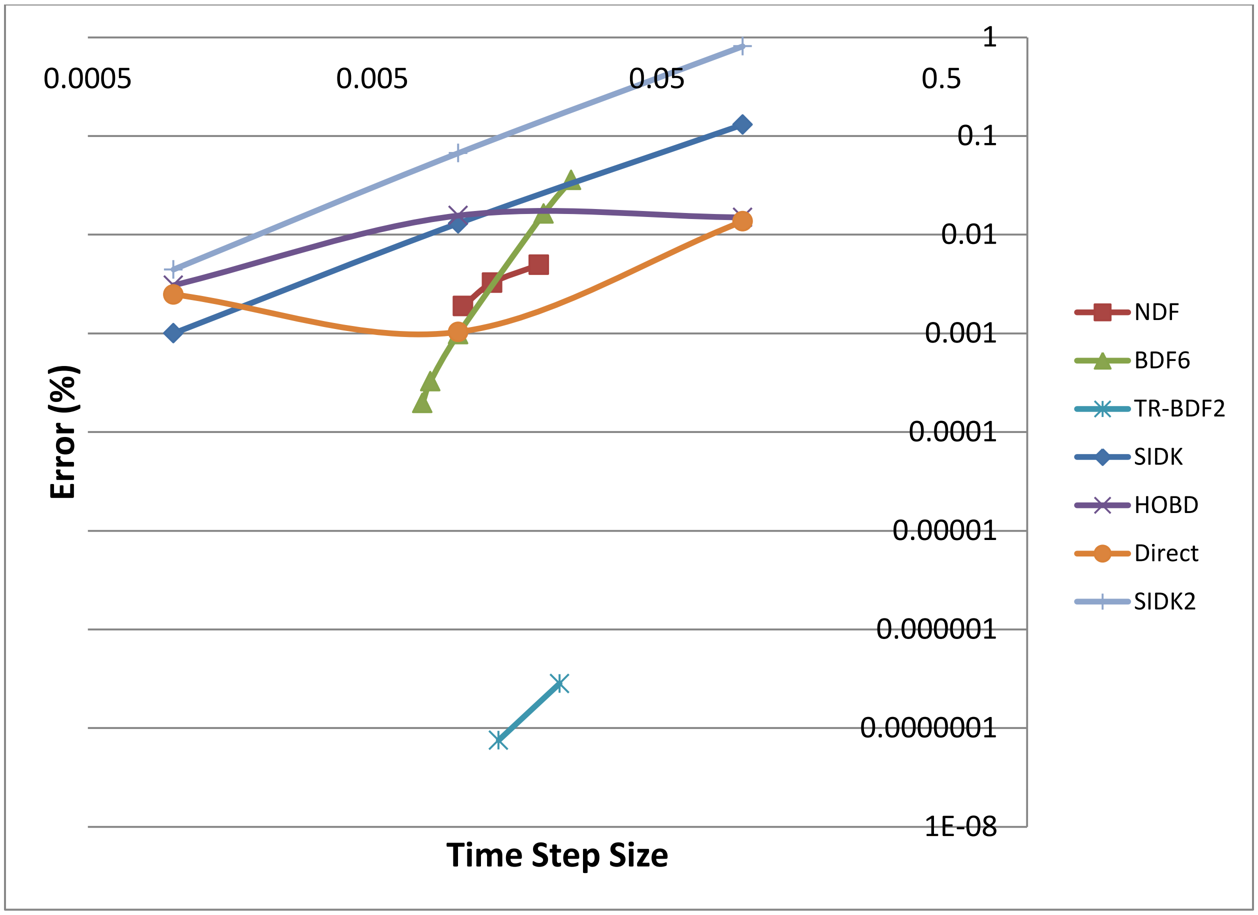

Figure 19 Convergence orders of various solvers

The efficiency of each solver is shown in Figure 20. Of all of the first order choices, the HOBD represents the best run time at coarse error tolerance, suffering no run time penalty vs. the SIDK method and giving lower errors at very coarse time step sizes. The SIDK method represents the best choice at intermediate error tolerances, giving slightly lower errors at the same run times as the other methods. At low error tolerances, the TR-BDF2 solver is by far the most effective solver, yielding extremely low errors (1E-7 %) with fairly coarse time steps. However, the TR-BDF2 method is slow due to the multi-stage, multi-step process and is therefore not preferred at larger error tolerances. The TR-BDF2 method is demonstrated to be a second order method.

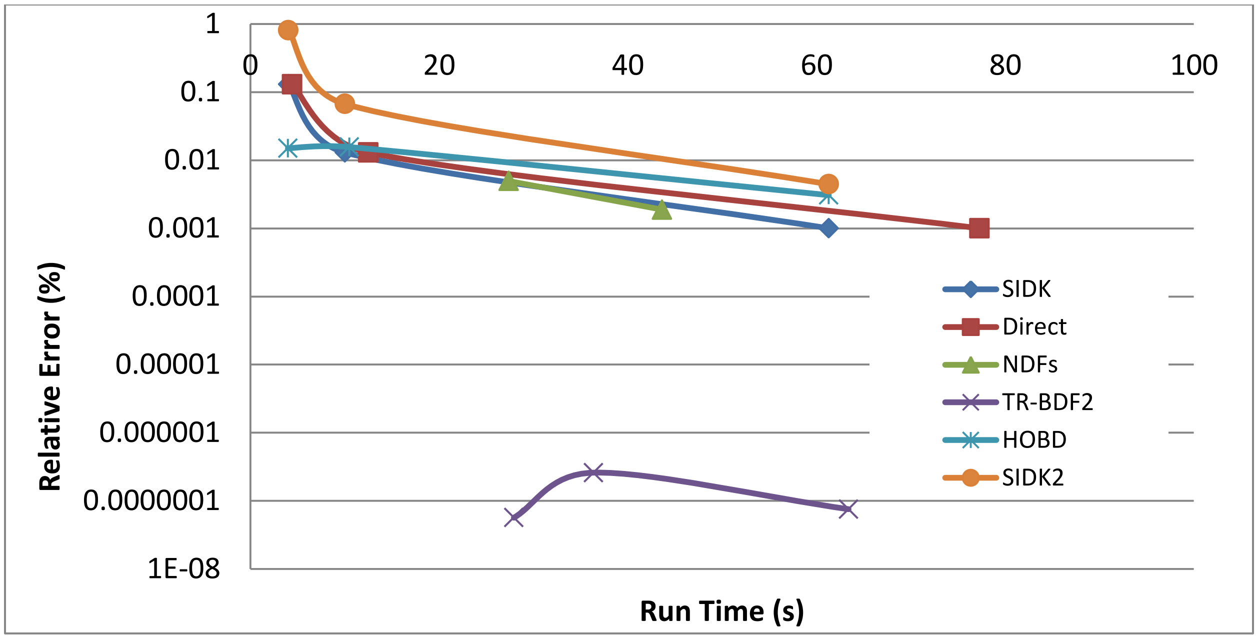

Figure 20 Performance of various solvers

5.3 TWO-DIMENSIONAL BWR BENCHMARK WITH THERMAL FEEDBACK (ANL-14-A1)

The 14-A1 (or LRA benchmark (12)) was selected to demonstrate the feasibility of the method for complex geometries with thermal feedback as applied to light water reactors. The purpose of the two-dimensional, two-group diffusion code was to test the SIDK method to compute the time and spatially dependent power distribution with thermal feedback and compare with fully implicit direct kinetics, the HOBD method, as well as reference solutions (12) (6) (14). The developed program used first-order finite difference spatial discretization, as described in section 2.3.1 . Temperatures were represented with a first-order backward difference formulation, while cross sections were represented with a first-order forward Euler formulation. In this section, a problem description subsection will be followed by a series of subsections describing results and analysis.

5.3.1 LRA Problem Description

Some of the benchmark parameters are presented below. The geometry, cross sections, and thermal feedback model for the benchmark are included in Figure 21, as well as in Table 17 and Table 18, followed by the thermal feedback specifications and equations 5.1 through 5.3.

$\nu = 2.43$
$v_1 = 3*10^7$ cm/s
$v_2 = 3*10^5$ cm/s
Dimensions = 165 cm X 165 cm (Each block in Figure 21 is 15 cm by 15 cm)
Initial Mean Power in Fuel Regions = $1*10^{-6}$ W/cm^3
Axial Buckling (B^2) = $1*10^{-4}$ for all regions, both energy groups
Material 6 is the control blade

Perturbation

$\Sigma a_2(t) / \Sigma a_2(t) = 1 - 0.0606184*t$ $t<=2$ s

0.8787631 $t>2$ s

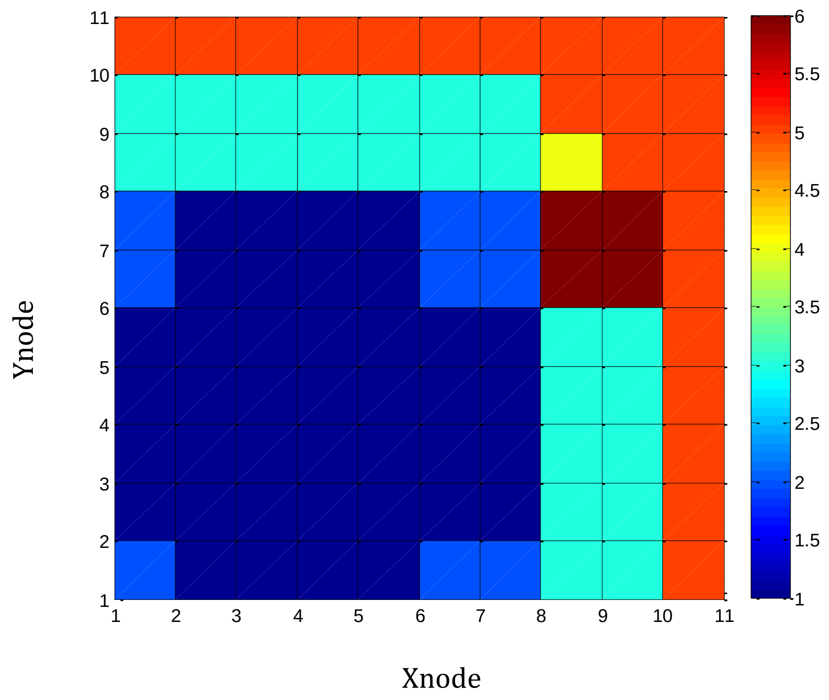

Figure 21 Geometry and region numbers for LRA benchmark

Table 17 Cross Sections

Region	Material	Group i	D_i (cm)	Σa_i (cm^{-1})	$\upsilon\Sigma f_i$ (cm^{-1})	$\Sigma_{1\rightarrow 2}$ (cm^{-1})
1	Fuel 1 with rod	1	1.255	0.008252	0.004602	0.02533
		2	0.211	0.1003	0.1091	
2	Fuel 1 with-out rod	1	1.268	0.007181	0.004609	0.02767
		2	0.1902	0.07047	0.08675	
3	Fuel 2 with rod	1	1.259	0.008002	0.004663	0.02617
		2	0.2091	0.08344	0.1021	
4	Fuel 2 with-out rod	1	1.259	0.008002	0.004663	0.02617
		2	0.2091	0.073324	0.1021	
5	Reflector	1	1.257	0.0006034	0	0.04754
		2	0.1592	0.01911	0	

Table 18 Delayed Neutron Data

Group	β_i	λ_i (s^{-1})
1	0.0054	0.00654
2	0.001087	1.35

$\alpha = 3.83*10^{-11}$ Kcm^3

$\gamma = 2.034*10^{-3}$ $K^{-1/2}$

$\varepsilon = 3.204$ 10^{-11} J/fission

$$\alpha\left[\Sigma_{f_1}(x,t)\phi_1(x,t) + \Sigma_{f_2}(x,t)\phi_2(x,t)\right] = \frac{\partial}{\partial t}T(x,t) \qquad \text{Equation 5.1}$$

$$\Sigma_{a_1}(x,t) = \Sigma_{a_1}(x,0)\left[1 + \gamma(\sqrt{T(x,t)} - \sqrt{T(x,0)}\right] \qquad \text{Equation 5.2}$$

$$P(x,t) = \varepsilon\left[\Sigma_{f_1}(x,t)\phi_1(x,t) + \Sigma_{f_2}(x,t)\phi_2(x,t)\right] \qquad \text{Equation 5.3}$$

5.3.2 LRA Steady-State Results

The eigenvalue computed with the developed two-dimensional two-group diffusion code (designed to test the SIDK method) was within 1 pcm of that reported in the benchmark. The control blade worth was within 6 pcm of that reported in the benchmark. The steady-state normalized powers were within a maximum difference of 3% of the benchmark, with an average of less than 1% error, as were the normalized powers at 0.4 s into the transient. In order to make extensive mesh refinement possible, an iterative solution strategy was adopted. The iterative package utilized for these solutions was Trilinos (65). The iterative Aztec solver from Trilinos was employed for the solution of the linear systems involved. The Aztec solver uses a Krylov (66) subspace method with an incomplete LU (ILU) factorization pre-conditioner. A comparison of the eigenvalues is shown below in Table 19, while the normalized steady-state powers are shown below Figure 22 and differences in Figure 23. Employing the Trilinos iterative solution strategy greatly sped up the run times, and was what made the comparison with the direct method possible. The SIDK method still outperformed the direct method (in run-time) by a factor of 2, which validates the computational advantage of using this method. Table 19 clearly demonstrates that the eigenvalue of the code had converged to the correct solution.

Table 19 Eigenvalue Comparison

Code	Mesh Size	Eigenvalue	Pcm Difference
CUBBOX	15 X 15 cm	0.99663	Ref
SIDK	15 X 15 cm	0.99790	127
SIDK	5 X 5 cm	0.99722	59
SIDK	2.5 X 2.5 cm	0.99683	20
SIDK	.9375 X .9375 cm	0.99664	1

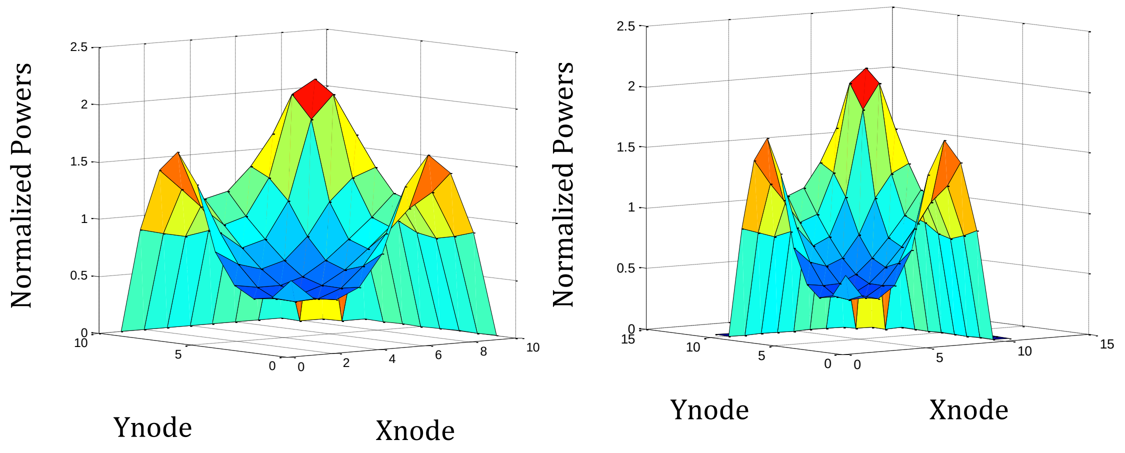

Figure 22 LRA steady-state case (SIDK program left, benchmark right)

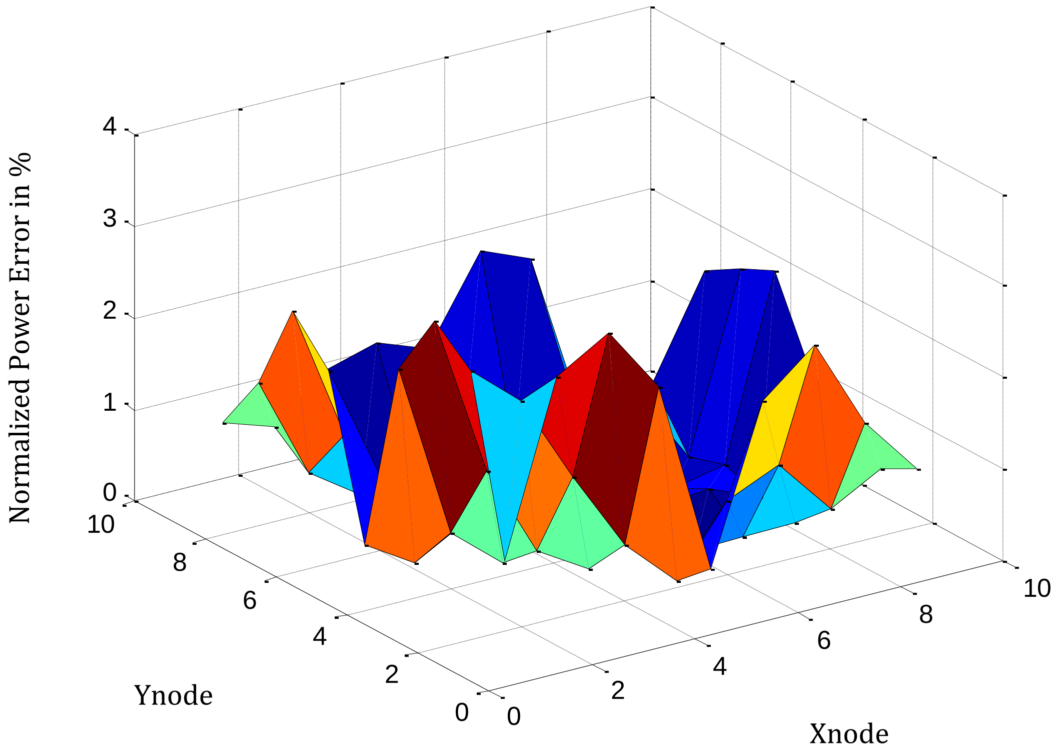

Figure 23 Percent Error of SIDK Diffusion code vs. CUBBOX Benchmark

5.3.3 LRA Transient Results

The transient represents a control blade drop, which is initiated by decreasing the thermal absorption cross section of the rodded fuel in four adjacent assemblies. This results in a super prompt critical transient. Below, in Figure 24 through Figure 28 are power and temperature comparisons of SIDK vs. nodal codes CUBBOX and CONQUEST as well as the MOC code DeCart. We can see the differences in the powers at t=0.4 s (Figure 25) are very similar to those shown in Figure 23. Note that detailed comparisons of power shapes as a function of time were not provided in Gehin (6). Table 20 compares methods within the previous work while Table 21 compares the SIDK results to the results from the literature (6) (12) (14).

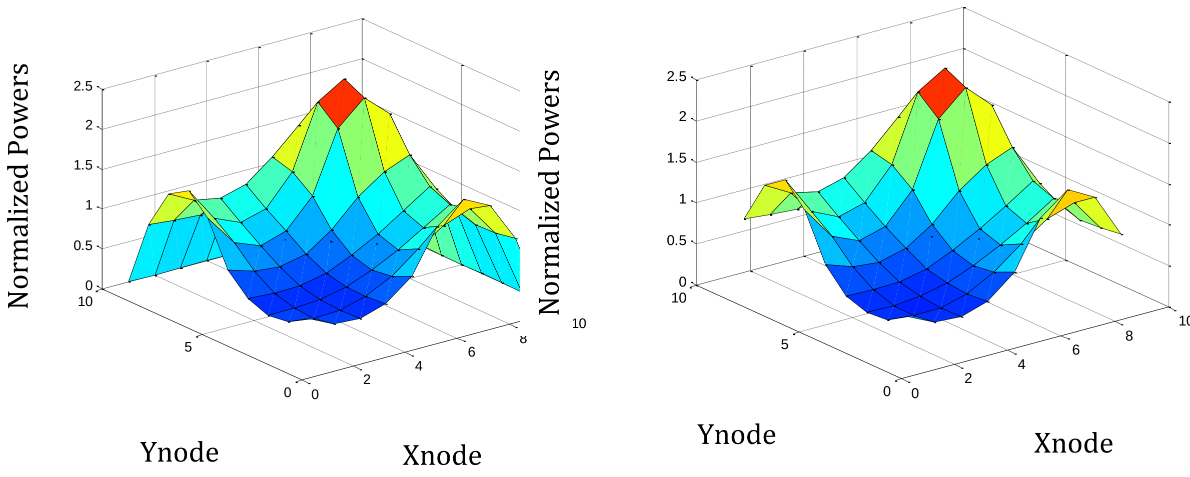

Figure 24 Normalized Power, t=.4s (SIDK left, CUBBOX right)

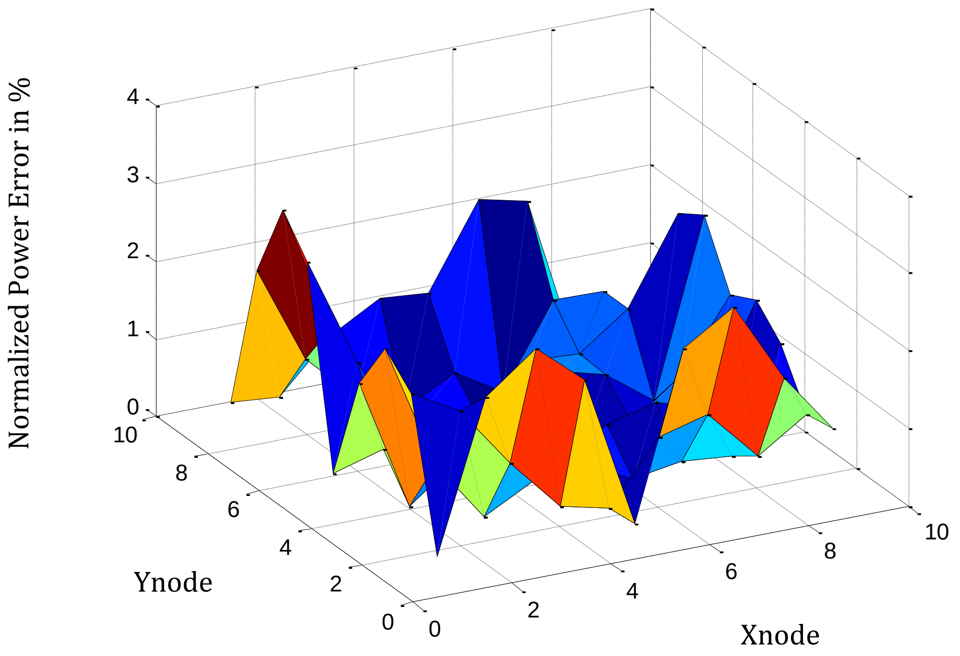

Figure 25 Percent Error of SIDK Diffusion code vs. CUBBOX benchmark, t=.4 s

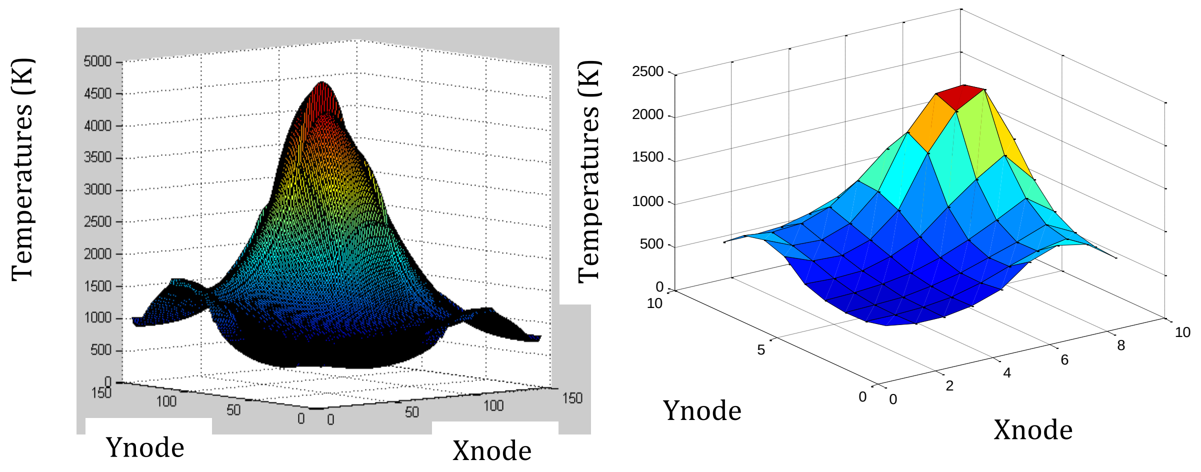

Figure 26 Temperature, t=2s (SIDK left, CUBBOX right)

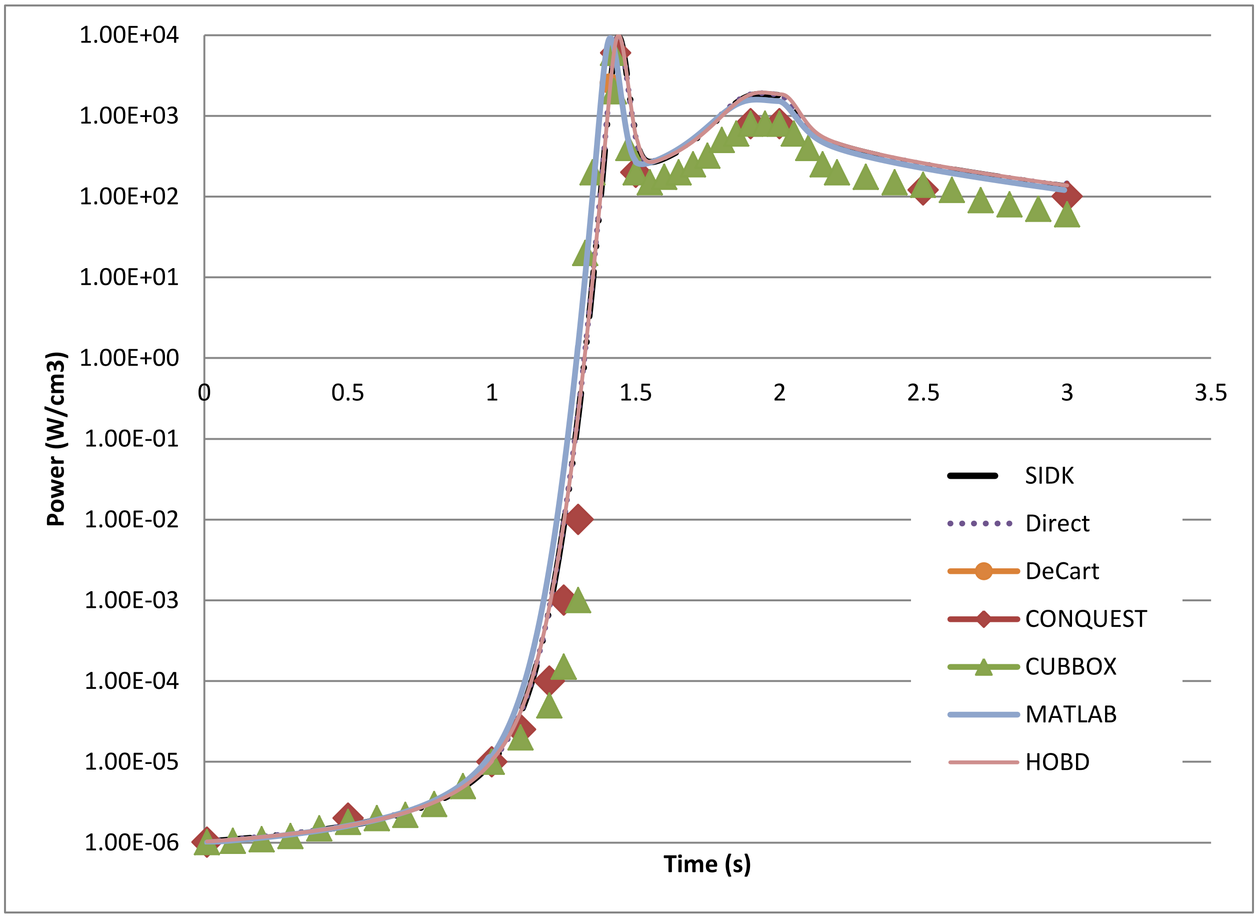

Figure 27 Power vs. Time

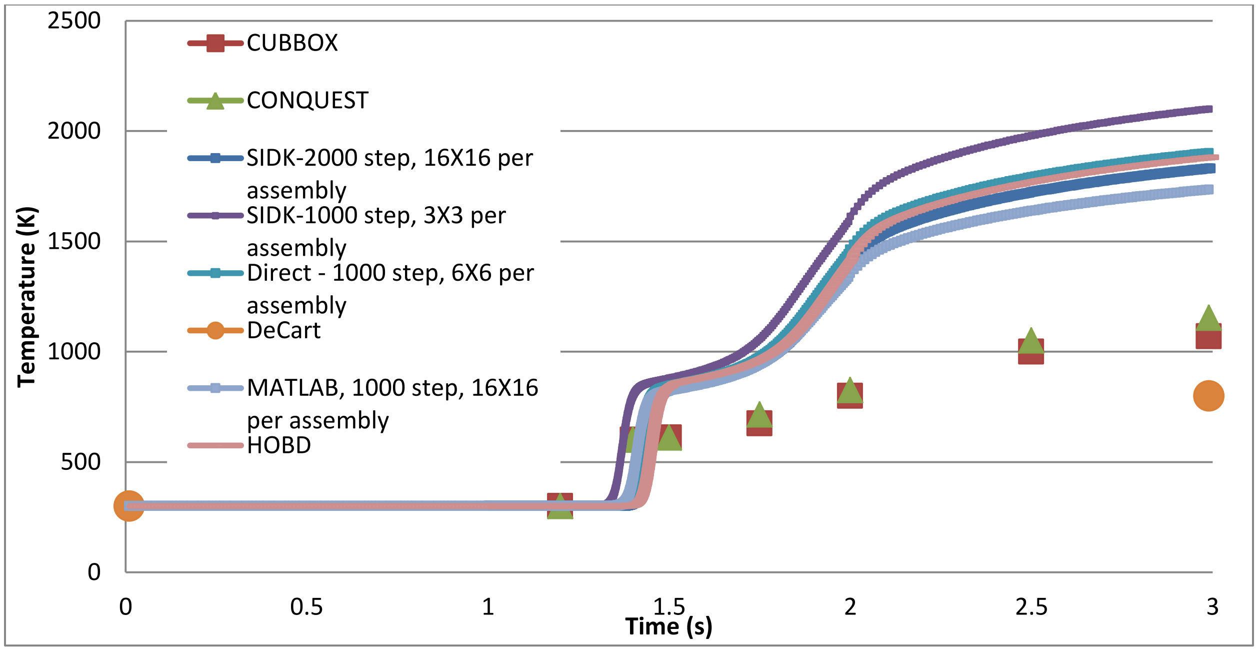

Figure 28 Temperature vs. Time

Table 20: Speed and Accuracy for LRA Benchmark Comparison, SIDK versus Direct Kinetics

Code	Direct Kinetics (Reference)	SIDK	HOBD	MATLAB (Independent)
Peak Power	9441 W/cm^3	9460 W/cm^3	9957 W/cm^3	9202 W/cm^3
Difference	N/A	0.2 %	5.58 %	2.53 %
Time	1.4395	1.44 s	1.44 s	1.4115 s
Difference	N/A	0.03 %	0.03 %	1.95 %
Average Temperature	1796.4 K	1800 K	1881.3 K	1733.7 K
Difference	N/A	0.2 %	2.19 %	3.5 %
Speed (s)	94 s	60 s	60 s	43 s
Speed-up	N/A	1.6 x faster	1.6 x faster	2.2 x faster

Table 21: LRA Benchmark Comparison, FMFD versus Nodal and Transport Methods

Code	SIDK	CONQUEST (Reference) (6)	CUBBOX (12)	DeCart (14)
Peak Power	9460 W/cm^3	5439 W/cm^3	5734 W/cm^3	2570 W/cm^3
Difference	~30 %	N/A	~ 10 %	~ 50 %
Time	1.44 s	1.438 s	1.421 s	-
Difference	0.1 %	N/A	~ 2 %	-
Average Temperature	1800 K	1154 K	1070 K	800 K
Difference	~30 %	N/A	~ 10 %	~ 30 %
Speed (s)	60 s	163 s	180 s	-
Speed-up	N/A	N/A	1.1 x slower	-

5.3.4 LRA Comparison with Literature Solution Discussion

For the benchmark calculations, the methods used in this work versus those reported in the benchmark were similar in nature, in as much as the benchmark calculations herein reported utilized a two-group, two-dimensional diffusion solver. However, the benchmark was designed to test coarse mesh or "nodal" methods, one of which utilized a sixth order polynomial Nodal Expansion Methods (NEM) to treat transverse leakage between large or coarse nodes. Other codes reported in the benchmark used similar types of higher-order expansions (12) (6). As such, the benchmark used one node per assembly, resulting in 15 cm X 15 cm cells. Initially, the program developed herein to test the benchmark used a first-order linear discretization, and was not designed to be a coarse mesh method. In addition, the initial program utilized dense matrix direct solvers, which made only limited mesh refinement possible. Two different direct solver packages were employed: a generic open source package, and the GNU Scientific Library Package (67). The GNU Scientific Library (GSL) implementation employed an LU decomposition, which was not optimized for the bandwidth of the problem. This implementation was further limited by the dense storage method utilized by GSL. The semi-implicit direct kinetics (SIDK) program used 15 X 15 cm mesh cells, or one node per assembly, as well as 5 X 5 cm mesh cells, or nine nodes per assembly, utilizing the direct solvers. The finer mesh necessary for the lower order discretization to reproduce the higher order nodal methods also results in a difference in the thermal feedback implementation, as there was now considerable thermal variation, and therefore cross section variation permitted within an assembly. Averaging the fluxes for each assembly to use assembly averaged temperatures had a few percent effect on the results, which increased as the mesh was refined, which is presented below.

At t=0s, the SIDK program is very consistent with the benchmark solutions. At t=0.4s, the SIDK method is still very consistent with the benchmark, where thermal feedback has not had an impact yet. At this time in the transient, the temperatures in the nodal codes as well as the diffusion program are nearly identical to 300 K. It appears that the boundary conditions are handled differently between the nodal codes and the fine-mesh finite difference (FMFD) code created to test the SIDK method. The initial differences (Figure 23) are quite low, but it can be seen that they are the highest close to the reflected boundary condition. Similarly, at t=0.4s, the highest differences (Figure 25) are near the reflected boundary conditions, suggesting there may be some small leakage approximation differences between the FMFD and the Nodal methods, but that overall the two methods agree well. Therefore, it appears that the difference in the two results (Nodal methods versus FMFD) is coming from a difference in the thermal feedback implementation. It is possible that the authors of the nodal codes interpreted the feedback model to mean removal cross section, rather than absorption cross section, which would lead to a substantially greater thermal feedback. It is also possible that some other relevant parameter is not being interpreted in the same fashion by the authors' of the nodal methods versus this work.

In order to verify the program developed, the first co-author on this work in conference publications (11) (68) Steven Hamilton wrote an independent verification program in MATLAB utilizing fine mesh finite difference. Though the program created by Hamilton used a different discretization (cell centered instead of face centered), it utilized the same kinetics method (SIDK). The agreement demonstrated below between the Hamilton MATLAB program and the SIDK C++ program written by the author is within 4 %, which is quite consistent with the agreement between the various nodal methods. Despite the discrepancies between the two FMFD codes and the Nodal methods, we can see that the general trends, shapes, and overall predictions of the SIDK program are still reasonable for such a severe transient, which results in a prompt supercritical configuration. In addition, the smaller mesh did produce values that were closer to the published values of a number of codes obtained from the benchmark and other references (6) (12).

Furthermore, a direct kinetics (fully-implicit) approach was also utilized in the SIDK test program. This approach yielded powers that were closer to the benchmark published values and within 0.2 % of those predicted by the SIDK method. Lastly, there has been some contemporary effort by others to reproduce these results, using the MOC code DeCart (14). Similar substantial differences between the nodal methods and DeCart were shown in that work, indicating that this problem, due the severity, is highly sensitive to the method selection, which is definitive in the sense that two independent methods applied by three independent authors had substantial differences from the benchmark. Therefore, on this transient where quasi-static methods are slower than direct methods, it is observed that the SIDK method still performs faster by roughly a factor of 2 with acceptable accuracy for kinetics results of within 0.2 %. Note that for this type of challenging problem, accuracy within ~5 % is deemed adequate (12) (6).

Above, in Figure 27 Power vs. Time, we can see the SIDK program captures the same trends and general results as the coarse mesh methods. However, Figure 28 illustrates that SIDK over-predicts the power, resulting in a large over prediction of average fuel temperature. This is due to the fact that for the adiabatic model used, there is no way for the fuel to cool off. Therefore, the temperature is essentially just the power integrated over time multiplied by a constant. The over prediction of power does appear to be due to differences between FMFD and Nodal Expansion Methods (NEM). However, given the similar differences observed in Table 20 from the DeCart approach (14), the SIDK method still performs comparably on this complex and highly severe LWR transient with thermal feedback, considering the errors resulting from the discretization and the spread of results seen in current attempts to duplicate benchmark results. This is particularly illustrated by the favorable results and run time comparison against the direct method, utilizing both direct and iterative solvers. Figure 28 further illustrates the homogenous temperatures for each assembly in the nodal codes, versus the heterogeneous temperatures within an assembly of the FMFD approach.

5.3.5 LRA Efficiency Analysis

In order to quantify the impact of the thermal resolution on temperature feedback, the FMFD SIDK code was altered so that each assembly would use an assembly averaged power in order to compute an assembly averaged temperature, which would then be used in the adiabatic thermal feedback model to update the cross sections. The errors in average fuel temperature, which is the best integral quantity over the transient due to the use of the adiabatic model, for various spatial and temporal discretizations for both the SIDK method and direct are presented in Figure 29. It is observed that the direct kinetics 352 time step mesh, at fine spatial meshes, has a lower error than the 1000 step SIDK method, demonstrating that for this problem, the direct method outperforms the SIDK method also due to the low number of precursors for this problem, which is two. The 1000 step fully implicit direct method is used as the reference solution.

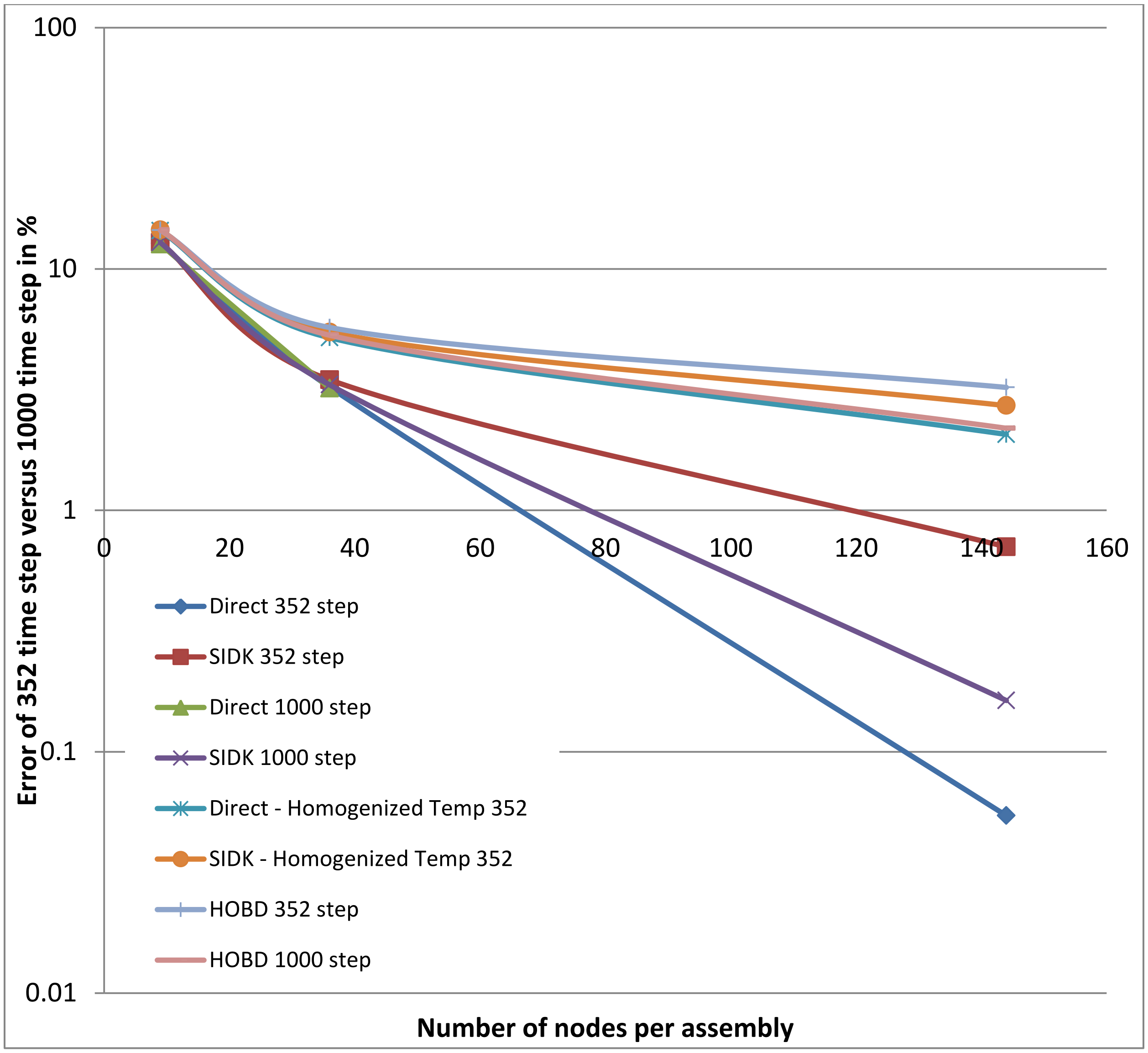

Figure 29 Errors resulting from the 352 time step mesh versus the reference solution from the 1000 time step mesh in final average fuel temperature

Though at intermediate time meshes (352 step), fully-implicit direct kinetics produces lower error and run-times than the SIDK method at 1000 time steps, direct kinetics was found to be subject to large oscillations with a suitably large time step (92 step). In addition to this, the SIDK method using either 352 steps or 1000s steps outperforms the direct method as medium spatial resolutions (at 36 or less nodes per assembly). The SIDK method, with its explicit representation of power over the time step, utilizing the power at the beginning of the time step, was found to be more accurate versus the HOBD method on this problem, as the error due to the explicit update of cross-sections led to higher power rises per time step with the semi-implicit representation of power.

Chapter 6 TIME-DEPENDENT NEUTRON TRANSPORT

This chapter serves to examine the resolution of angle and neutron energy of the time-dependent Boltzmann equation in high fidelity, massively parallel radiation transport solvers able to represent an arbitrarily high number of neutron energy groups, where up to forty-four energy groups are used. The first problem examined is an infinite homogeneous medium, which serves to quantify temporal errors resulting from various time step sizes as well as to ensure that the method as coded into the Denovo radiation transport package is numerically identical (within machine precision) to the method as implemented in MATLAB (for the infinite homogeneous problem only). The second problem examined is the 16-A1 benchmark, previously examined in Section 5.1 . This section serves to quantify the advantage of discretizing angle via the discrete ordinates approach versus the neutron diffusion approach where angle is not treated. In addition, this problem serves to quantify the errors introduced by the Denovo implemented SIDK method. The third problem serves to illustrate the potential fidelity and scalability of the method by modeling a 3X3 “mini-assembly” control rod ejection using forty-four neutron energy groups in fully heterogeneous geometry, where the problem and transient is comparable to the current state of the art with the exception of the significant increase in neutron energy group fidelity and within pin temperature distirbutions offered in this dissertation work (59) (36).

6.1 INFINITE HOMOGENEOUS MEDIUM (IHM) PROBLEM

The SIDK method was derived and introduced in section 3.1.4 following the descriptions of the factorization and the direct methodologies. The SIDK method shares some features of both of these approaches, while featuring its own approximations. There are two main approximations: the first approximation involves replacing the angular flux with the scalar flux moments in the source term, initially using only the zeroth moment, or neglecting the temporal flux derivative altogether, both of which are considered in this work. It is demonstrated that for fast reactors with high velocities or for thermal reactor undergoing mild transients this assumption is quite valid, generally introducing errors that are 1 % or less. The second approximation entails treating the precursor source term explicitly, which enables a sequentially solvable form of the coupled transport and precursor equations, and allows the transport equation to be solved separately from the precursors for the direct method. The computational advantages of this method include the ease of implementation in high-fidelity massively-parallel transport codes and comparatively low memory requirements (59), as well as faster speeds than direct methods using the same time discretization.

In order to confirm convergence as well as the order of convergence of the SIDK method to an exact solution, an infinite homogenous medium problem was selected that could be solved exactly using point kinetics, as there is no shape dependence. In the creation of this verification program, an off-the-shelf academic program was leveraged (MATLAB) which explored several new aspects of point kinetics for verification and solution approaches.

In order to obtain an exact solution for the two-group infinite homogeneous medium problem, point kinetics can be used. However, without solving the adjoint problem, each energy group must be represented explicitly to obtain the exact solution. The difference in using multiple energy groups in the point kinetics equations is the addition of one ODE for each additional energy group, which allows explicit representation of each group's mean generation time. This difference primarily shows up in the prompt jump portion of the transient; however, the difference is minor after the prompt jump.

The SIDK method was coded into MATLAB for an infinite homogeneous benchmark problem. This method, which is the method used for the spatial kinetics calculations in Denovo, was compared to a multi-group point kinetics formulation to ensure that the method was numerically stable and error bounded. The method is convergent when the time-dependent change in flux term is included, which it is in this problem; some other problems considered however, have a minimum error due to the neglect of this pseudo absorption term. Therefore, in addition to the verification of the coding of the method in Denovo and the demonstration of the linear convergence of the method, an examination of the importance of the inclusion of the time-dependent change in flux is also included here. The parameters considered in this problem are consistent with the mean generation time of a fast reactor. It is demonstrated that in the fast reactor case, neglecting the time dependent change in flux term introduces a small error, particularly for short time periods. This is relevant in the results presented in the section 5.1 , for which the impact of neglecting the time dependent change in scalar flux is also considered. The SIDK method used time steps ranging from 0.1 seconds to 1e-5 seconds.

6.1.1 IHM Problem Description

The problem considered is a two-dimensional evaluation of an infinite homogenous medium. There is no material variation and all boundary conditions are reflective. The perturbation considered is a 69 cent step in reactivity. The parameters are shown below in Table 22 and Table 23, for which this reactivity perturbation results in a power increase of a factor of three. The Denovo case is compared to MATLAB for the same conditions.

Table 22: Point Kinetics Parameters

Parameter	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
Beta	2.42734e-4	1.45207e-3	1.33159e-3	2.91368e-3	1.03430e-3	2.5233e-4
Lambda	1.27172e-2	3.16469e-2	1.16552e-1	3.13889e-1	1.38265	3.82153

Table 23 IHM Cross Sections

Group	Sigma F	Sigma T	Sigma g->g	Sigma g->g’	Velocity
1	7.4518e-3	1.849e-1	1.777e-1	2.085e-3	5.402e8 cm/s
2	1.1061e-2	3.668e-1	3.537e-1	0	9.191e7 cm/s

6.1.2 IHM Results and Analysis

The power rise comparison is shown below in Table 24, which effectively shows identical results (using all significant figures the difference was 2.5E-8 %). The Denovo computed powers for the SIDK method were compared to the Matlab computed powers to ensure that the method was coded properly into Denovo, yielding errors all below 2E-8 %, which is effectively the tolerance of the solvers utilized. The results of the various time steps versus the two-group point kinetics solution solved using the eigenvalue decomposition (which is exact for the infinite homogeneous problem) is shown in Figure 30, while the error of the semi-implicit method with and without the time-dependent change in angular flux is shown in Figure 31.

Table 24: Infinite Homogenous Medium Power at t=10 s Comparison

Code	Final Power	Percent Difference
Denovo	1426.442992	0.0 %
MATLAB	1426.442992	N/A

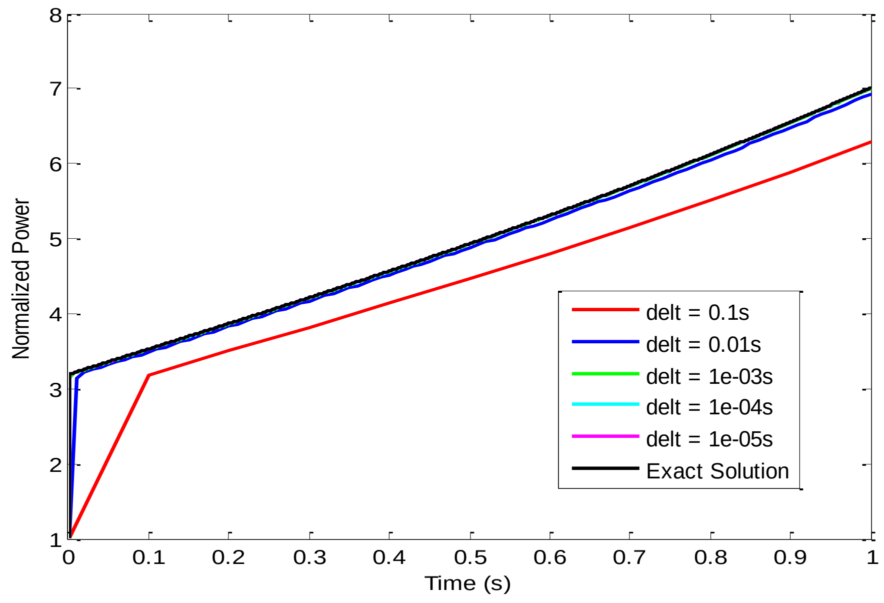

Figure 30 Semi-Implicit Direct Kinetics with dϕ/dt (red largest time step, pink smallest time step) versus Two-group point kinetics using eigenvalue decomposition, which is exact (in black)

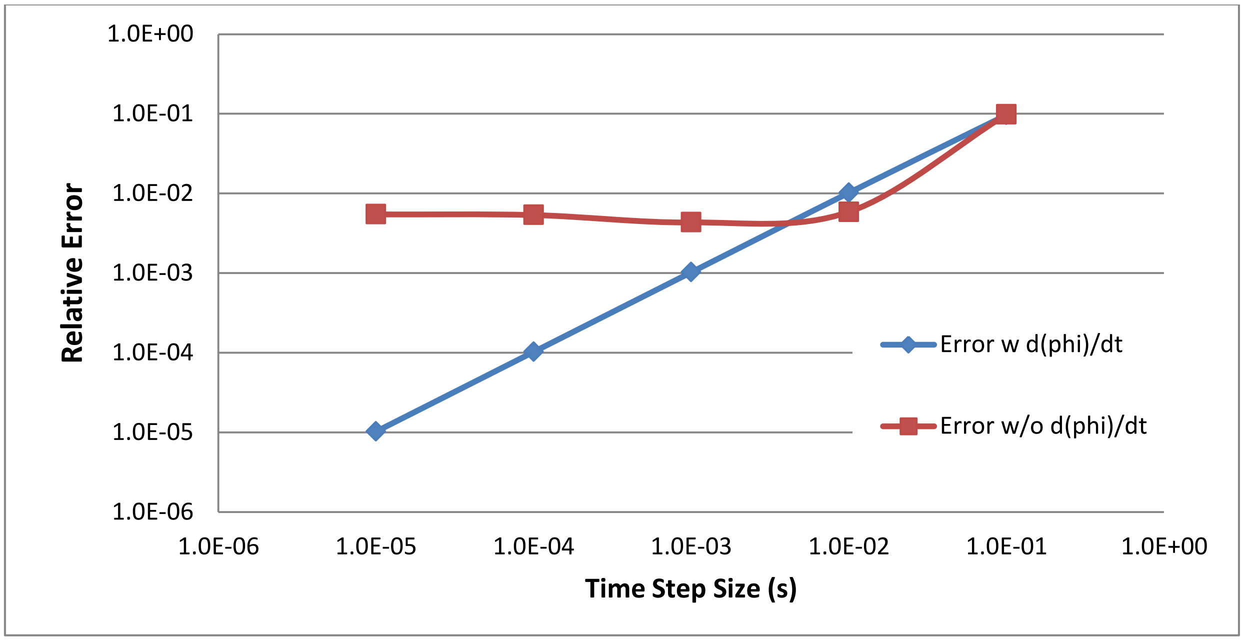

Figure 31 Relative error of semi-implicit decoupled direct kinetics with and without inclusion of dϕ/dt

Figure 30 clearly illustrates that the semi-implicit direct kinetics (SIDK) method converges to the multi-group point kinetics solution (solved exactly). It is clear from Figure 31 that the error decreases linearly with time step size, and that the method is linearly convergent when the time-dependent change in angular flux is included. It is clear when this term is neglected that there is a minimum error with time refinement that comes from treating the initial power rise, which takes a fixed amount of time, as an instantaneous jump. It is demonstrated that the magnitude of this error is proportional to the mean generation time of the neutrons; therefore, the prompt jump approximation is generally acceptable for fast systems, while generally inacceptable for thermal systems. The normalized powers of the SIDK method using the coarse time step size of 0.01 s with and without the inclusion of $\partial\phi/dt$ as well as the exact multi-group point kinetics utilizing the eigenvalue decomposition method are presented in Figure 32, while the error as a function of time resulting from the coarse time step size of 0.01s or the neglect of $\partial\phi/dt$ is shown in Figure 33.

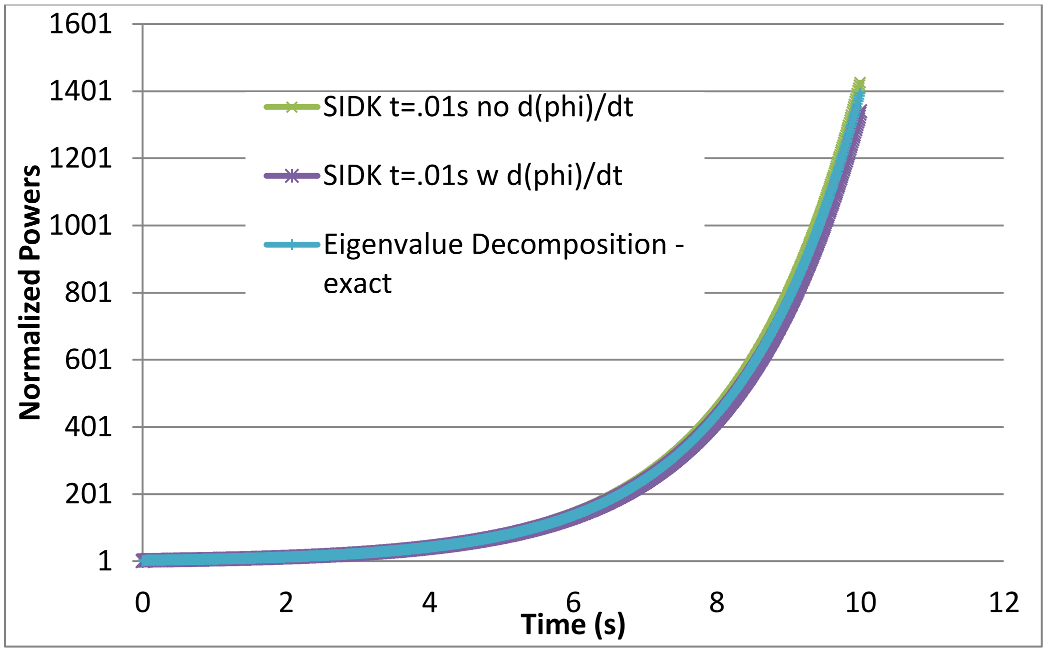

Figure 32 Infinite Homogeneous medium test problem

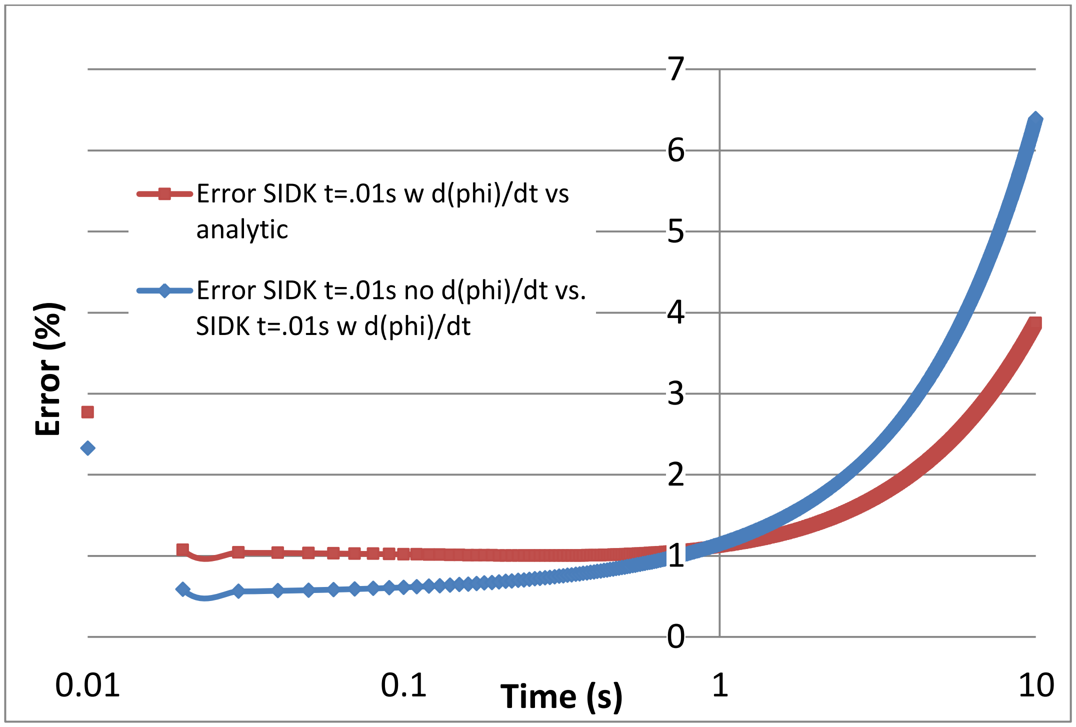

Figure 33 Infinite homogeneous benchmark error of coarse time step size vs. analytic solution

6.2 Transport - One-Dimensional Fast Reactor Benchmark

This section will serve to verify the spatial kinetics implementation in Denovo as well as demonstrating the impact of using a neutron transport solution, where angular discretization is treated, by revisiting the same benchmark problem that was in Chapter 5 with the use of the Denovo radiation transport solver as opposed to the SIDK time-dependent diffusion code.

6.2.1 16-A1 Transport – Steady-State Results

The first step in the spatial kinetics process is the eigenvalue calculation. As Denovo is only two- or three-dimensional, the problem was run with reflecting boundary conditions on the top and the bottom to convert the problem to be effectively 1D. The eigenvalue calculated was within 1 pcm of the 1.000198 reported in the benchmark, which is substantially closer than the 650 pcm difference from the finest mesh case of the diffusion program. In addition, the comparison of diffusion versus the transport benchmark and Denovo serves to illustrate the advantages of transport solutions. It is demonstrated that due to poor treatment of interfaces in diffusion approximations to the transport equation, both the eigenvalue and spatial flux distribution calculated in the diffusion implementation suffer spatial errors that are not removed under mesh refinement. The spatial flux comparison is shown below in Figure 34, which illustrated that all of the fluxes were within 1 % of the benchmark. As relative error is plotted, this quantity is generally highest on the boundaries of the problem, as that is where the flux is the lowest. The flux comparison with the diffusion program from section 5.1 is shown in Figure 35, quantifying the error typically introduced in diffusion approximations, particularly at the material boundaries.

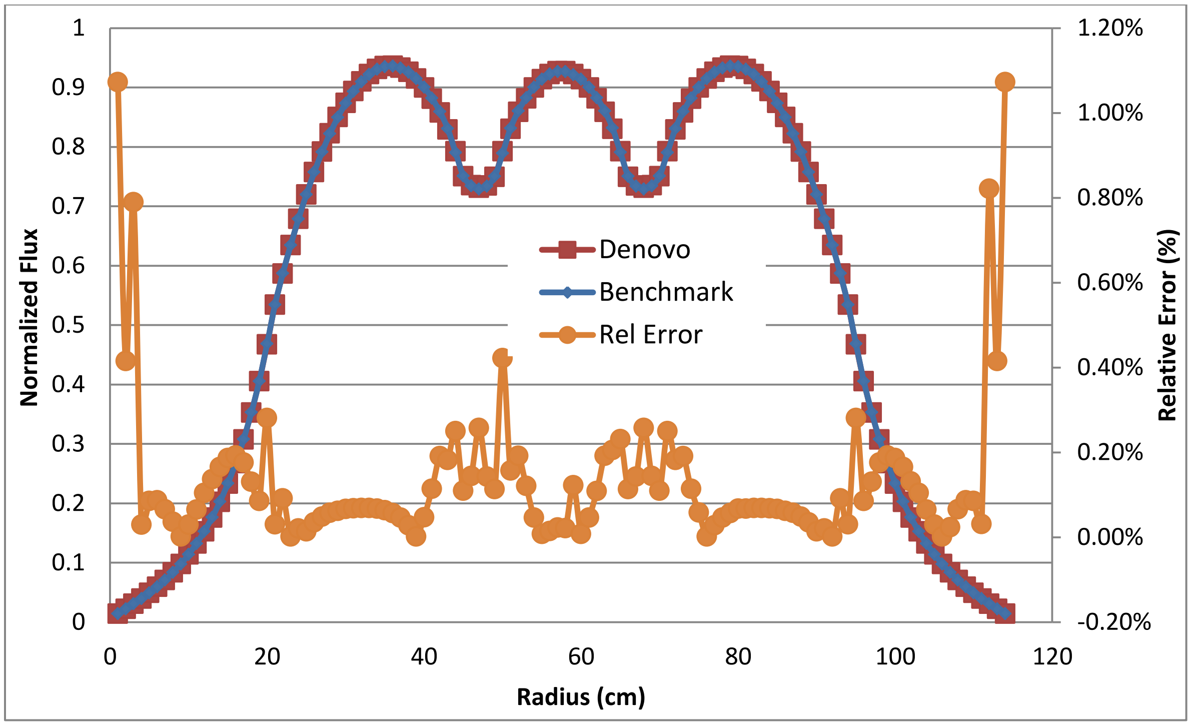

Figure 34 Denovo vs. ONEDANT group 1 (fast) flux solution

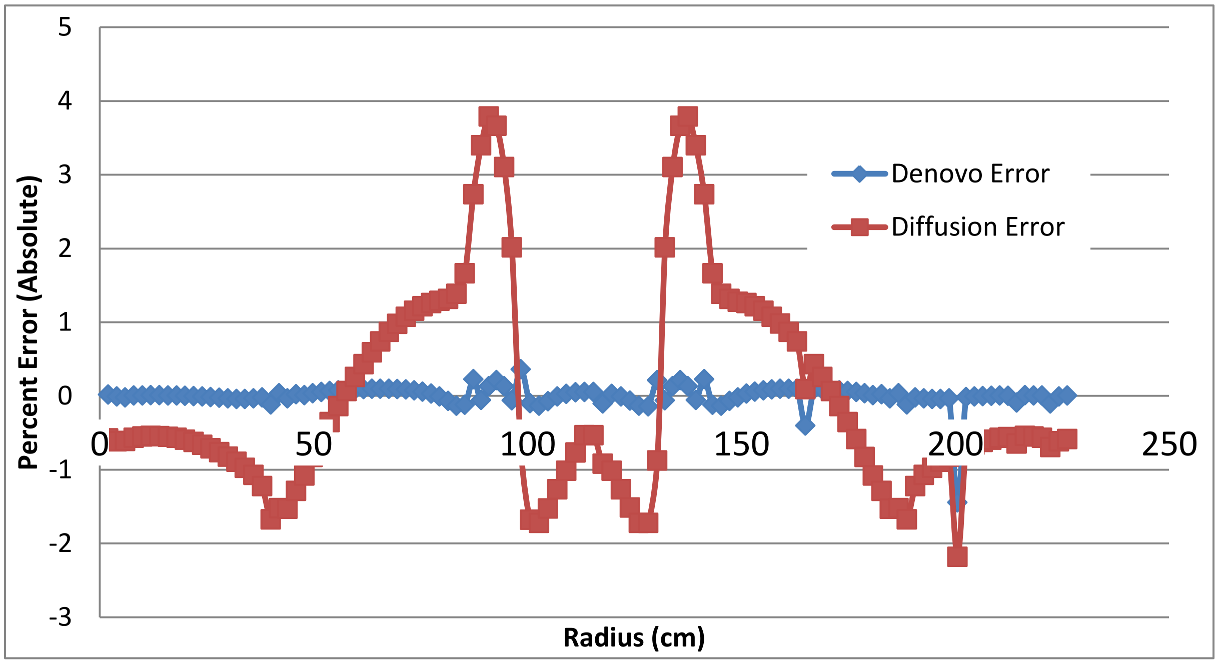

Figure 35 Errors at t=0s versus 16-A1 Benchmark in %, Group 1 Flux for Denovo and Diffusion

6.2.2 16-A1 Transport – Transient Results

To continue the development of the spatial kinetics capability for the AMP-Denovo multi-physics framework, the next step was to obtain the time-dependent fluxes, starting with the benchmark solution at t=0.01s, where this simulation neglected the time-dependent flux term. The impact of using the isotropic approximation in Equation 5.11 has been demonstrated in other works (59) (14). The impact of including this term is shown above in section 6.1.2 as well for this problem in section 5.1 . The initial fluxes come from the steady-state solution shown in Figure 34. The perturbation is a change in material densities in two of the fissile regions, which constitutes an increase in the fission cross section in the leftmost region while simultaneously decreasing the fission cross section in the rightmost region by 5 % each. The time-dependent flux solutions are run as fixed source problems. The prompt jump or first time step is run using steady-state spatially dependent precursors as the fixed source. All subsequent time steps are run using analytically determined space and time-dependent precursors. The comparison of the Denovo fluxes using the SIDK method versus the benchmark flux at the only time-step included in the benchmark where the error (of the benchmark codes) was deemed acceptable was at t=.01 s after the perturbation, which is shown below in Figure 36, which exhibit a very reasonable agreement. All of the perturbed fluxes were within 5 % of those given in the benchmark solution.

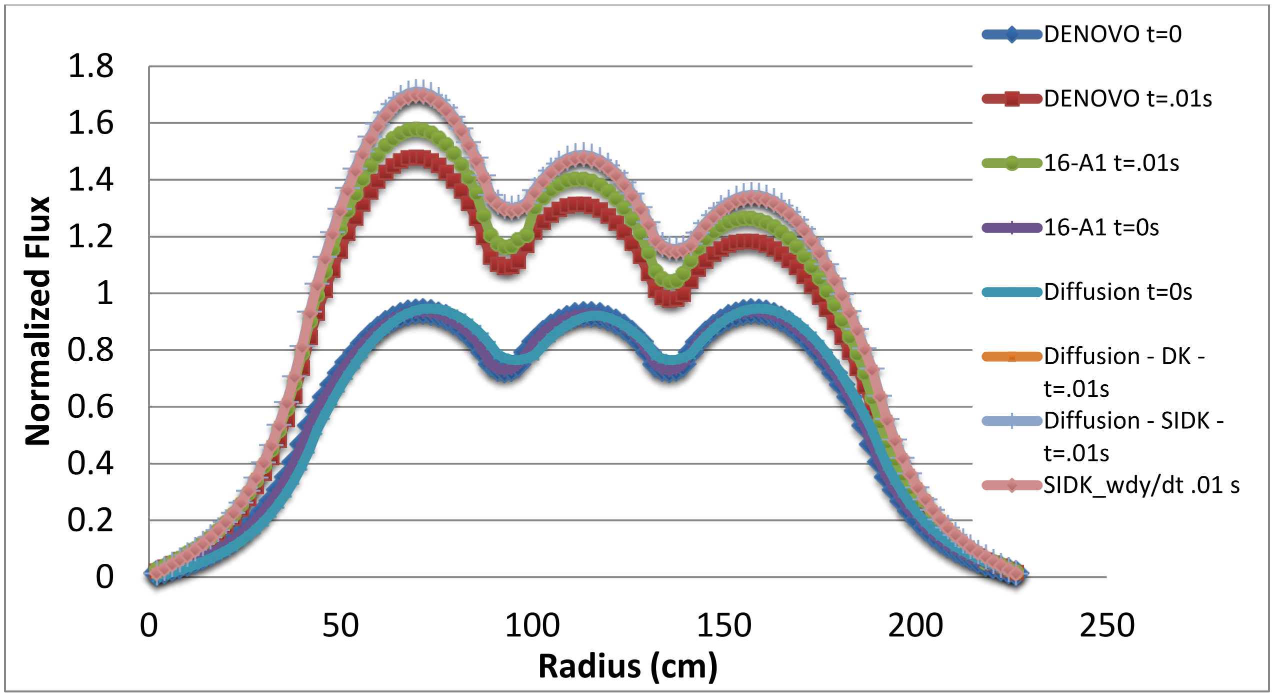

Figure 36 Transport - Group 1 Steady-State and Perturbed Fluxes

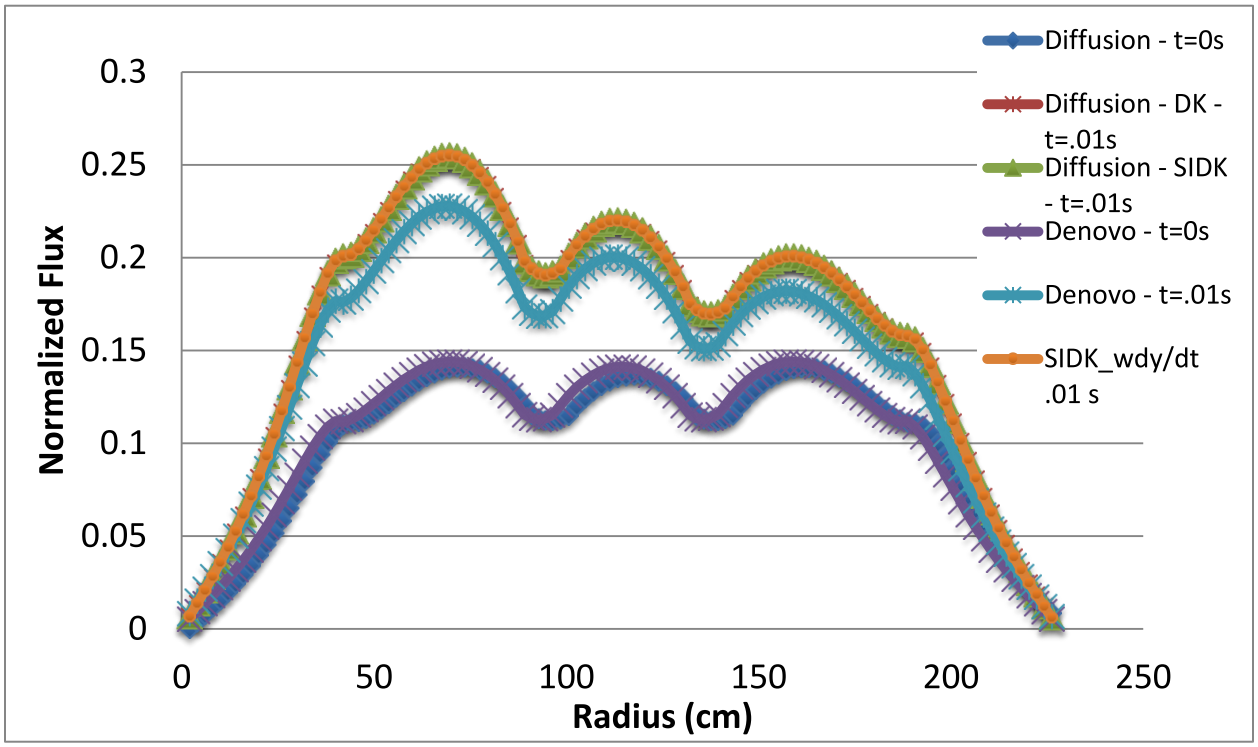

Figure 37 Transport - Group 2 Steady-State and Perturbed Fluxes

6.2.3 16-A1 Transport - Analysis

In order to tie the BRISC package, implemented in AMP, to the Denovo SIDK computed powers, as well as compare with the point kinetics results reported in (12), the BRISC package was utilized within AMP to compute the time-dependent spatially averaged powers for the 16-A1 problem. The successful completion of this objective further verifies the point kinetics capability implemented in AMP for fast and thermal systems, which demonstrates its readiness to be utilized as a point kinetics package for a quasi-static spatial kinetics implementation within the AMP multi-physics framework. In order to calculate the amplitude weighting function for quasi-static kinetics, point kinetics is used. The parameters for the point kinetics solutions come from the adjoint flux collapsed over energy and angle, which were taken from the point kinetics parameters presented by Goluoglu (13). The comparison of the amplitude function using a Runge-Kutta multi-step predictor corrector algorithm, the BRISC package as implemented in AMP and the 16-A1 benchmark are shown below in Figure 38, which exhibit excellent agreements. The benchmark utilized a quasi-static implementation, for which a reactivity coefficient is calculated for each time step.

The SIDK implemented method in Denovo uses the perturbation from the benchmark, which neglecting the time-dependent change in flux, causes Denovo to over-predict the powers at very short time scales. After 0.001 s however, the Denovo results are in good agreement (within 5 %) with the point kinetics results reported in the benchmark, where the results reported are from the adjoint weighted point kinetics functions, or the aforementioned amplitude function.

As noted previously, there were some discrepancies in the results reported in the benchmark. For example, as quasi-static kinetics uses the amplitude function as a weighting function for the magnitude of the integrated spatial powers, the integrated spatial powers from the benchmark should exactly match the amplitude function. While the agreement reported in the benchmark is not exact, it is evident that the spatially integrated magnitude of the benchmark solution at 0.01s does closely match the amplitude function from the benchmark, as well as the AMP and RK45 computed powers. In addition to this discrepancy, as the codes used in the benchmark deviated from one another by more than 2 % after 0.01s (which was the maximum deviation allowed in the benchmark problems (12)), this was the only time step that the spatially dependent powers were available for. The Denovo results between 1 and 2 s are slightly lower than the point kinetics results from the benchmark, which was also the case for the spatially integrated values from the benchmark versus the point kinetics results in the benchmark at t=0.01s. As mentioned, this benchmark was primarily chosen for several reasons: it is a neutron transport based benchmark, which is not common for kinetics benchmarks, it allowed a point kinetics comparison with the amplitude function, which offered a chance to benchmark the AMP point kinetics, and it is a fast reactor, while many other kinetics benchmarks are thermal reactors. Given the discrepancies of the codes utilized in the benchmark, the 5 % agreement exhibited by Denovo with the benchmark results is deemed to be reasonable.

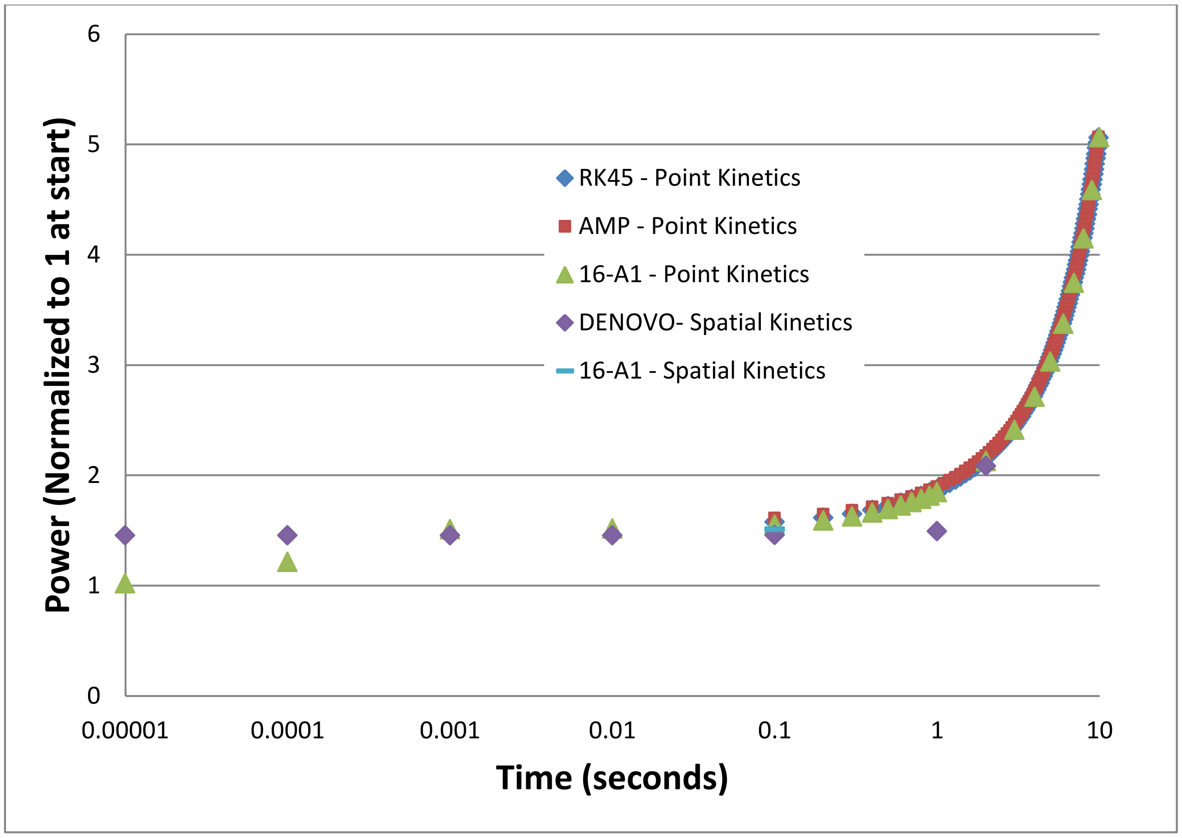

Figure 38 RK45 vs TDTort Amplitude

6.3 Halden Reactor

In order to demonstrate the parallel, high-fidelity, heterogeneous geometrical treatment, arbitrarily high number of neutron energy group advantages offered by implementing the demonstrated non-intrusive stable and robust SIDK method within steady-state radiation transport solvers internally coupled in a multi-physics framework, a heterogeneous reactor problem that had been verified and validated with the AMP thermo-mechanics code was chosen for a kinetics problem (17) (69) (70) (16). The reactor problem considered is the Halden research reactor, which is a heavy boiling water reactor (HBWR) located in Norway near the Swedish border. The reactor is a heavy boiling water reactor with a maximum power of 25 MW thermal, with a water saturation temperature is 240 C (513 K) and the reactor operates at a pressure of 33.3 bar. A cross-section of the reactor is shown below in Figure 39 Cross Sectional Schematic of the Halden Reactor (61) (71). There are several advantages of using this problem as a demonstration of capability with some drawbacks. The advantages are that the Halden reactor is a well characterized research reactor, with many thermo-mechanics experiments including in-core thermal couples and detectors, experimental temperature data from SCRAM experiments, as well as post-irradiation experiments (PIEs). In addition to the experimental nature of the reactor, the thermo-mechanics benchmarks from the reactor have been widely used to validate fuel performance codes including FRAPCON (5) (72) (73) and AMP (15) (16), among others.

There were some drawbacks to this selection of demonstration problem, which were primarily centered on the fact that the reactor geometry was highly irregular; furthermore, it proved quite difficult to obtain detailed loading patterns for neighboring fuel rod and assembly information, due to the number of active test locations during any single test. Due to the highly irregular lattice, the geometry for the demonstration problem in Denovo was simplified to standard square pitch, as to make the demonstration problem more general in nature. As such, there are no experimental comparisons available for the results of the benchmark problem.

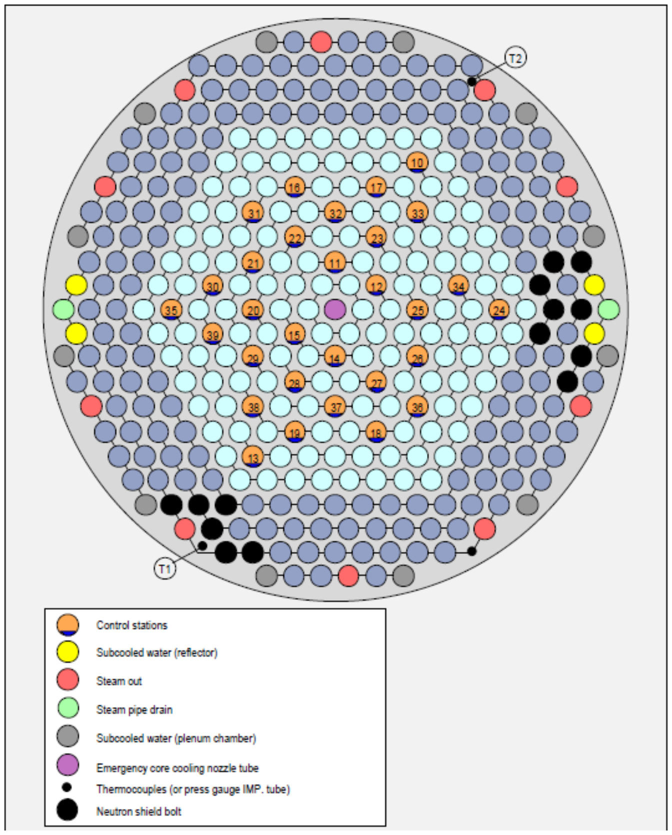
T2
T1
Control stations
Subcooled water (reflector)
Steam out
Steam pipe drain
Subcooled water (plenum chamber)
Emergency core cooling nozzle tube
Thermocouples (or press gauge IMP. tube)
Neutron shield bolt

Figure 39 Cross Sectional Schematic of the Halden Reactor Core

6.3.1 DENOVO - EIGENVALUE

The steady-state calculation begins with a data processing step, which consists of generating nuclear data with the SCALE (24) code, using the TRITON wrapper or the NEWT 2D discrete ordinates module. The next step is the generation of cross section libraries for steady-state (initial conditions) radiation transport using the CSAS-I module. Once these cross sections are generated, they are fed to an eigenvalue calculation using Denovo, which saves a cell wise spatial steady-state precursor source for spatial points owned by each processor.

In order to verify the geometry configuration for the initial conditions of the modified Halden problem, as there was no external verification or validation information available for the modified case, the NEWT 2D discrete ordinates module for SCALE was used. The results of the Denovo and corresponding NEWT solution are shown in Figure 40. The geometry for the control rod ejection problem is a 3 × 3 mini-assembly. This assembly consists of typical fuel pins from the Halden research reactor, which are ten percent enriched UO_2 fuel with Zirc-4 cladding. The fuel pins are approximately five mm in radius, with a pitch of 1.4224 cm. The Halden reactor is a Heavy-Boiling Water Reactor (HBWR) which is D_2O cooled and moderated. This geometry was modeled in NEWT, within SCALE, which employs a rigorous cross-section treatment able to provide multi-group cross sections that preserve reaction rates from 1D continuous energy data (from CENTRM) through the 238-group ENDF library. The CSAS-I module within SCALE was also used to generate cross sections for Denovo. The NEWT results were compared with the Denovo results, and the eigenvalues were 145 pcm different. Denovo yielded a k_{eff} of 1.06899, while NEWT yielded a k_{eff} of 1.07054. These differences are attributable to cross section processing differences between CSAS-I and CENTRM, used in NEWT. The visual differences in Figure 40 are due to the fact that Denovo uses a structured Cartesian mesh with volume weighted materials in each cell, while NEWT uses an unstructured extended step characteristic mesh which auto-refines to capture material boundaries. Mesh refinement mitigates these visual differences.

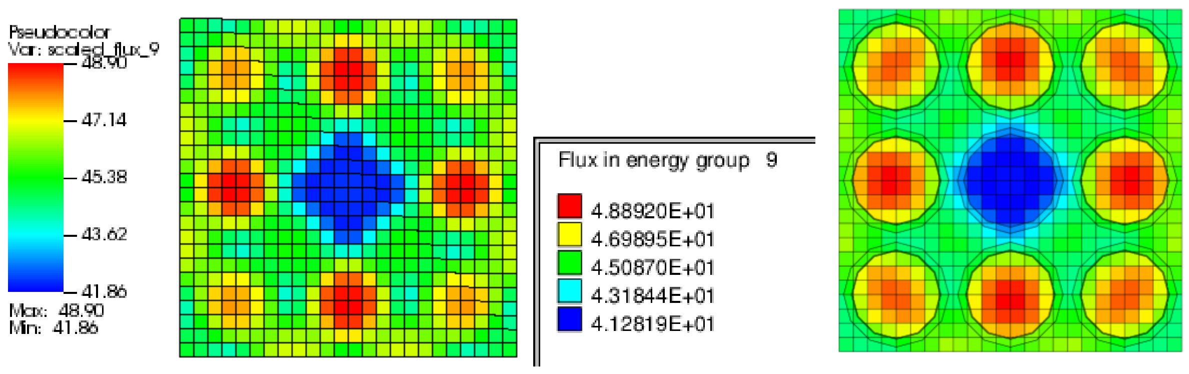

Figure 40. Denovo Multi-pin solution unnormalized fast flux (left) and NEWT unnormalized fast flux (right).

6.3.2 SCALING STUDY

A scaling study was performed on the University of Tennessee Nuclear Engineering Computational Cluster (NE-Cluster) (74) in order to determine the relative speed-ups to be expected in massively-parallel cases such as those run on supercomputers such as Jaguar (10) or Kraken (19), operated by the National Center for Computational Sciences and the National Institute of Computational Sciences, respectively. The NE-Cluster is a medium size, largely heterogeneous cluster consisting of thirty-one computational nodes possessing between four and forty eight cores each. In order to make a scaling study relevant, groups of identical cores were selected from the following groups, or queues:

- Gen3 – Core i7 nodes, 12 nodes in group, 8 cores each, 2.8 GHz each
- Gen4- Sandy-Bridge Core i7 nodes, 3 nodes in group, 8 cores each, 3.4 GHz each
- Super – A server node with four processors, each with 12 AMD cores. 48 cores total, 2.4 GHz each

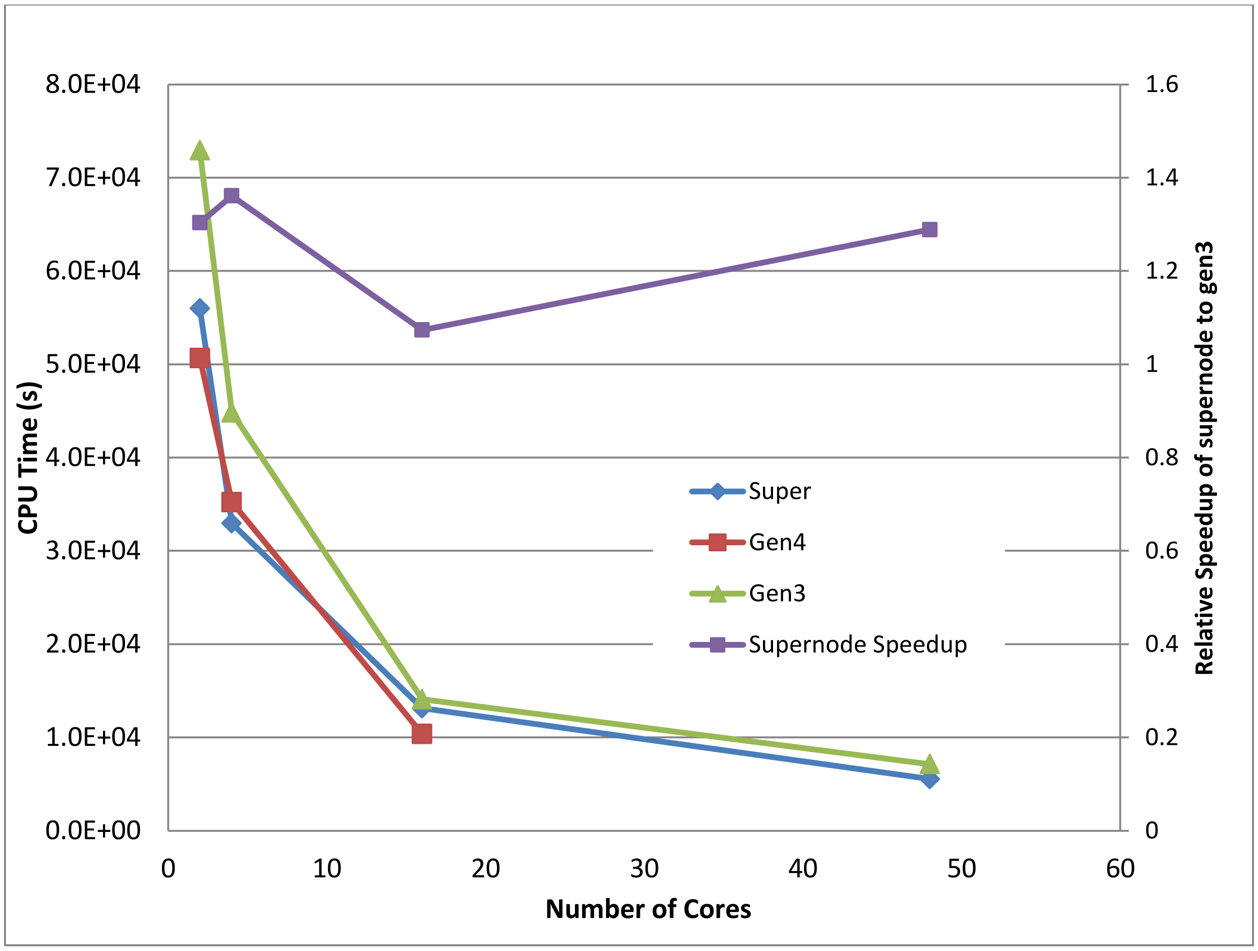

Figure 41 Denovo Scaling Study

We can observe that the speed-up of the case is roughly linear from two to sixteen cores. After sixteen cores however, the small spatial size of the problem (~80,000 mesh elements) is causing a limiting mesh decomposition, where in the 48 core case, each core is only receiving ~1,600 mesh elements. In other Denovo scaling studies, it has been determined that going below 2,500 mesh elements per core does not result in significant speed-ups as more cores are added, due to communication burdens, even if the cores are on the same processor. Communication overhead from Denovo is further illustrated in Figure 41 by the supernode computational speeds. Even though the supernode has slower cores than the gen3 nodes (2.4 GHz versus 2.8 GHz for the gen3 nodes), the supernode runs roughly 30 % faster than the gen3 nodes due to the reduced communication burden, which has been observed on the distributed nodes to be roughly ~40 % of the computational effort (spent in system calls for communication.) This study illustrates the computational advantages of parallelism for Denovo and acts as a guide for the appropriate spatial decomposition to use for varying processor/core counts when determining the number of cores appropriate for a particular spatial case as well as how large a case needs to be to warrant very large processor counts.

6.3.3 Multi-Physics - Thermal Sensitivity

The following process has been developed within the AMP framework. The initial step in the process involves generating delayed neutron data with the SCALE code. The next step is the generation of cross section libraries for the time dependent transport using the CSAS-I module. Once these cross sections are generated, they are fed to an eigenvalue calculation using Denovo within the AMP framework. The power distribution from the Denovo computation is mapped to the AMP code, which computes the temperature distribution. These temperatures are used in rings to update the cross section generation parameters in CSAS-I, which generates a new cross section library. These new cross sections are used to repeat the eigenvalue computation until it is converged (within 1 Kelvin) with the temperatures at which the cross sections were computed. In Figure 42 through Figure 44, the results of the two-way converged thermal solution to the Halden 3x3 problem, with the control rod removed, is depicted. The AMP input files were generated by utilizing the AMP input generator, using the parameters from the Denovo case. The Denovo inputs can be found in APPENDIX B.

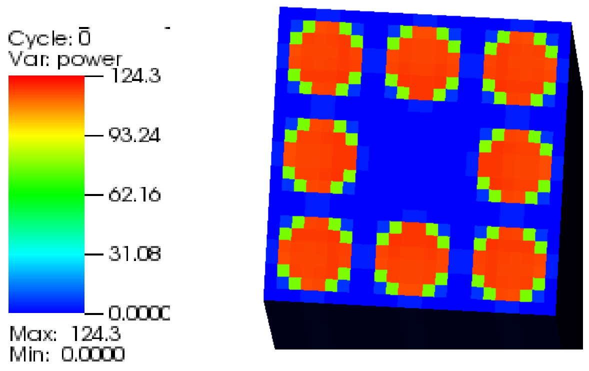

Figure 42 Denovo unnormalized power

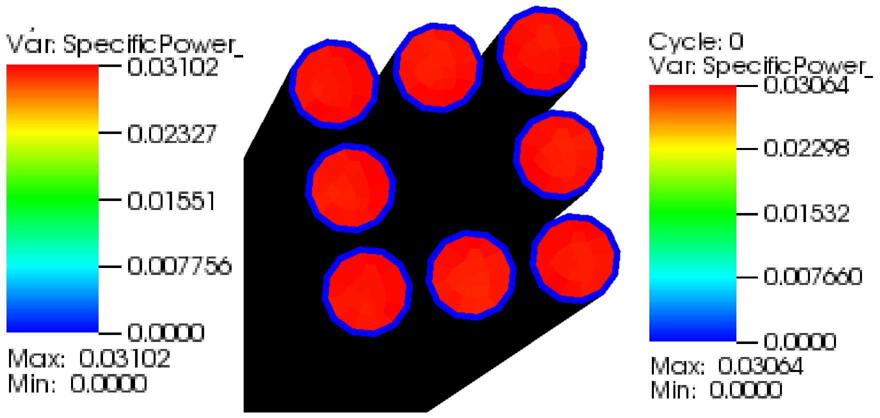

Figure 43 AMP normalized specific powers in W/g. Maximum power legend shown on left, minimum power legend on right

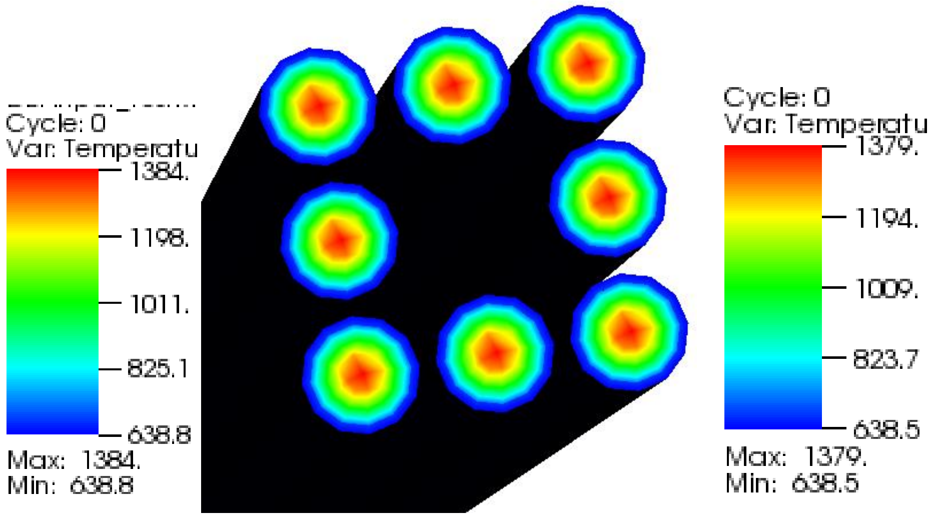

Figure 44 AMP pin temperatures (Kelvin) Maximum pin on Left, Minimum on Right

This level of within pin resolution is not currently present in any steady-state solutions, let alone transient solutions (68). In order to begin assessing the impact of the temperatures that were a result of the 3D calculated powers that were internally mapped to AMP, a sensitivity study was done. The first case uses cross sections which are all at 300 K, which represents no thermal calculation. The second case, or the average case, uses average pin temperatures, which is the current standard for high resolution transport based kinetics with thermal feedback (36). The third case uses within pin temperatures in rings, which represents a new level of fidelity (75). The sensitivity of the eigenvalues is shown below in Table 25.

Table 25 Eigenvalue Sensitivity to Cross-Section Temperatures

Resolution	None	Average	Ringed
K-eff	1.07658	1.02769	1.03001
Pcm off normal	0	4541	4326

In Table 25, predictably, having the temperatures of all materials perturbed several hundred degrees had a substantial negative impact on the eigenvalue, which is the expected result. However, when the temperatures were updated to represent the within pin distribution instead of the average, a change of over 200 pcm was observed. This is a fairly substantial change in the eigenvalue, which agrees well with preliminary related efforts (75). The reason for this shift is the combination of material self-shielding in the outer ring of the pellet that now has a lower temperature, which will further elevate fission cross sections. The higher temperature region in the center was already seeing a lower flux, so the fact that it now has lower chance to fission is mitigated by the increased cross sections in the higher flux outer ring.

6.3.4 ENERGY-GROUP SENSITIVITY

One of the major advantages offered by the Denovo radiation transport package over contemporary three-dimensional discrete ordinates time-dependent transport solvers (such as TDTORT **(13)**) other than its modernism and parallelism is the fact that it can represent an arbitrarily large number of neutron energy groups. In order to demonstrate the relevance of this addition, a sensitivity study was done on the 44-group library used to see the differences in eigenvalue and reactivity caused by using either a different number of energy groups or different cross-section processing routines for the same numbers of energy groups. To carry out this evaluation, a customized 8-group cross section library was prepared using the SCALE code suite using two different cross section processing options as well as the more traditional two group structure. The results of this study are shown below in Table 26.

Some background on the collapsing scheme used is needed in order to understand the large changes in eigenvalue observed in the collapsed libraries. The cross section collapsing mechanism in NEWT uses the following methodology:

- The eigenvalue and system fluxes are solved for using the fine-group cross section library
- The fine-group fluxes are collapsed using the <u>system average</u> flux
- Reaction rates are therefore only preserved for the <u>system average</u> fluxes.

Therefore, the spectral effects of each material are lost in the collapsing process (i.e. the spectrum should be softer around the guide tube where the water is due to moderation, and the spectrum should be harder in the fuel pins relative to the system average). This loss of spectral resolution lends itself to the large variability in eigenvalue shown below (76). As for the variation seen in the BONAMI cross section processing, BONAMI uses the Bondarenko method, which relies on the Narrow Resonance approximation, which is best suited to fast reactor applications. As the system under study is a thermal reactor, the Bondarenko method (using no Dancoff factor corrections) is not the ideal choice of cross section processing methods; however, its inclusion illustrates that impact of different cross section processing choices (77).

Table 26 Eigenvalue and Reactivity Sensitivity

Cross Section Processing	Number of Energy Groups	k-eff	pcm difference to reference	perturbed k	Reactivity (cents)	Reactivity % diff to reference
Continuous Energy (CENTRM)	44	1.08400	Reference	1.08455	7.60390	Reference
Continuous Energy (CENTRM)	8	1.10551	2151.3978	1.10604	7.34873	3.35580
Continuous Energy (CENTRM)	8	1.11730	3330.6138	1.11783	7.22224	5.01926
Bondarenko Method (BONAMI)	2	1.14351	5950.9196	1.14413	8.53003	12.17972

It can be observed from Table 26 that while the selection of cross section library had substantial impacts (up to 6000 pcm on eigenvalue), the impact on reactivity was considerably smaller, a maximum of around 10 %. However, in a three dollar prompt supercritical type transient, such as the LRA benchmark in section 5.3 , a 12 % difference in reactivity would correlate to a ~36 cent difference, which was considerably larger than the difference between the developed two-group two-dimensional FMFD code and the Nodal code CUBBOX (which was 6 pcm).

6.3.5 THREE-DIMENSIONAL CONTROL ROD EJECTION

The transient process using the SIDK method with Denovo begins with a data processing step, similar to the eigenvalue calculation, which consists of generating nuclear data with the SCALE (24) code, using the TRITON wrapper or the NEWT 2D discrete ordinates module. The next step is the generation of cross section libraries for steady-state (initial conditions) and time-dependent transport using the CSAS-I module. Once these cross sections are generated, they are fed to an eigenvalue calculation using Denovo, which saves a cell wise spatial steady-state precursor source for spatial points owned by each processor. This steady-state precursor source (in combination with the eigenvalue) is used to compute the perturbed state of the system, as in a material change caused by a control rod ejection, which then saves a cell-wise spatial time-dependent precursor source for spatial points owned by each processor. The analytical calculation of precursor source allows the use of a variable time stepping strategy over a number of time steps, which was demonstrated in section 6.2 .

The demonstration problem for the SIDK method in the AMP-Denovo multi-physics framework is a heterogeneous three-dimensional control rod ejection. The problems considered thus far have largely been benchmark problems, which for legacy methods employed, involved using homogenized regions employing two neutron energy groups. In consistency with modern approaches (59) (14), the SIDK method developed for the AMP-Denovo framework is applied in heterogeneous geometry in a high performance computing application, where heterogeneous refers to the resolution of pellets, gap, and clad utilizing many neutron energy groups with within-pellet thermal distributions and high performance computing refers to a large, parallel simulation using both multiple cores per compute node and many compute nodes. The control rod ejection problem involves multiple axial regions which represent a truly 3D problem, as the reactor already has inherent 2D nature with cylindrical fuel pins in a square lattice, and the control rod being removed axially at some finite speed. Therefore, the Denovo input was modified to include a second axial region in which the control rod was present in the upper portion of the assembly which would be withdrawn as well as a third region where the control rod would remain. The original Denovo input, in section 6.3.1 , did not include a control rod, as the comparison of eigenvalue was done with two-dimensional NEWT code, which due to the aforementioned cylindrical pins in a square lattice, would necessitate the control rod being fully inserted or fully withdrawn, for consistency. In order to facilitate faster run times, the 44 group ENDFB 5 library was used. A 24x24x112 mesh was used for the eigenvalue solutions, with a non-uniform axial mesh that was finer around the control rod. From this steady-state run, the steady-state precursor values were generated for each spatial region. The thermal flux (2 group collapsed) from the initial conditions of the problem is shown in Figure 45 at different axial heights, which clearly illustrates the three-dimensional nature of the problem. In each slice shown in Figure 45, the x-y variability of the problem is present due to the fact that there are cylindrical fuel pins within a square lattice. In the three slices, we can see moving from the lowermost slice to the uppermost slice that the flux sharply decreases due to the presence of the control rod, where all three plots are on the same scale. The Denovo inputs for these cases can be found in APPENDIX B.

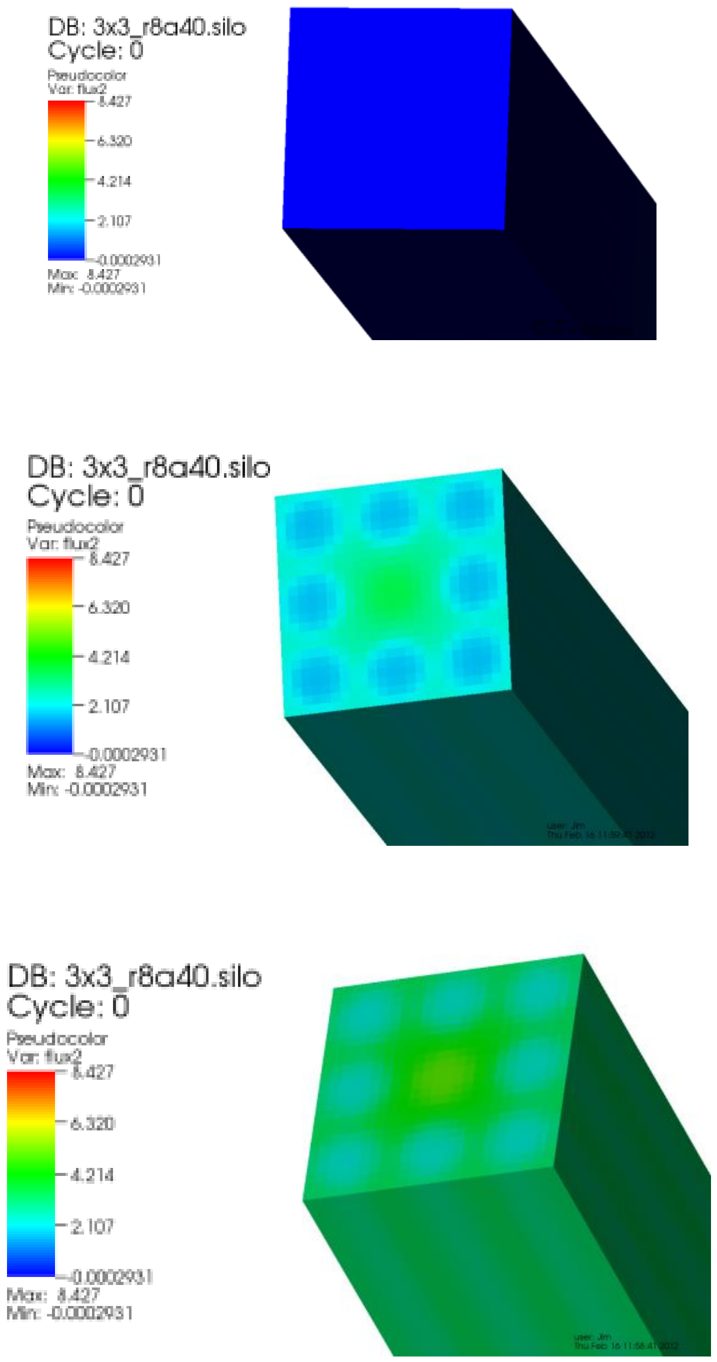

Figure 45 Initial Conditions Thermal Flux (unnormalized) at 3.8, 3.0, 2.5 m from top to bottom

The full control rod ejection case uses a 136 (axial) by 24 (x) by 24 (y) mesh with 44 energy groups (from AMPX via CSASI). The eigenvalue case is run first in parallel where each processor writes out its own source file that is specific to the mesh elements that it owns. The fixed source run reads in these source files, using the same number of processors as the eigenvalue run. This limits communication between processors, but creates an input-output (IO) burden that could be better handled by a script. The fission source in the fixed source is implicitly converged at each step, which is the most stable and accurate way of capturing shape changes. The energy fidelity is quite high compared to most kinetics methods which are two or four group, typically (6).

The rod ejection problem introduces a 6.6 cent reactivity insertion caused by a 4 cm control rod movement, which occurs over 0.01 s. The accompanying power rise was modeled using point kinetics which yielded a 0.5 % difference from the point kinetics vs. Denovo. The Denovo results are compared to point kinetics after 0.1 s, similar to the 16-A1 Argonne Benchmark problem. These results are shown in Figure 46. Due to neglecting the time-dependent angular flux term, this case represents essentially the prompt jump approximation. The next step of the problem involves the use of the analytic function to determine the precursor source. This is the same process that is used for all time steps beyond the second, demonstrated for the infinite homogenous medium problem. The results of the power rise from this function are shown in Figure 47, which can be compared to the point kinetics results shown in Figure 46.

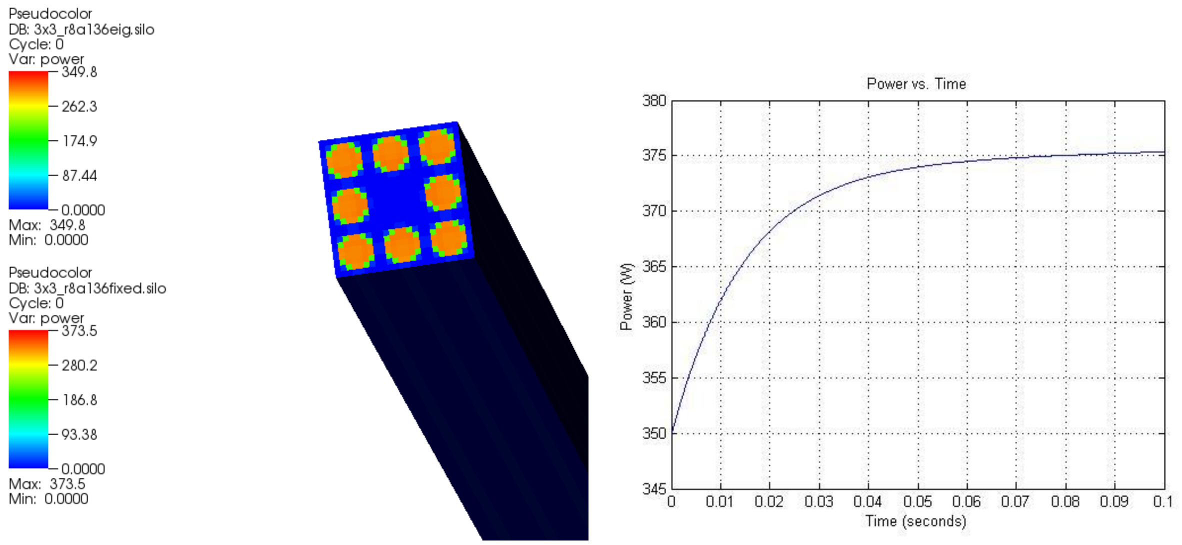

Figure 46 Perturbed Halden case vs. Point Kinetics

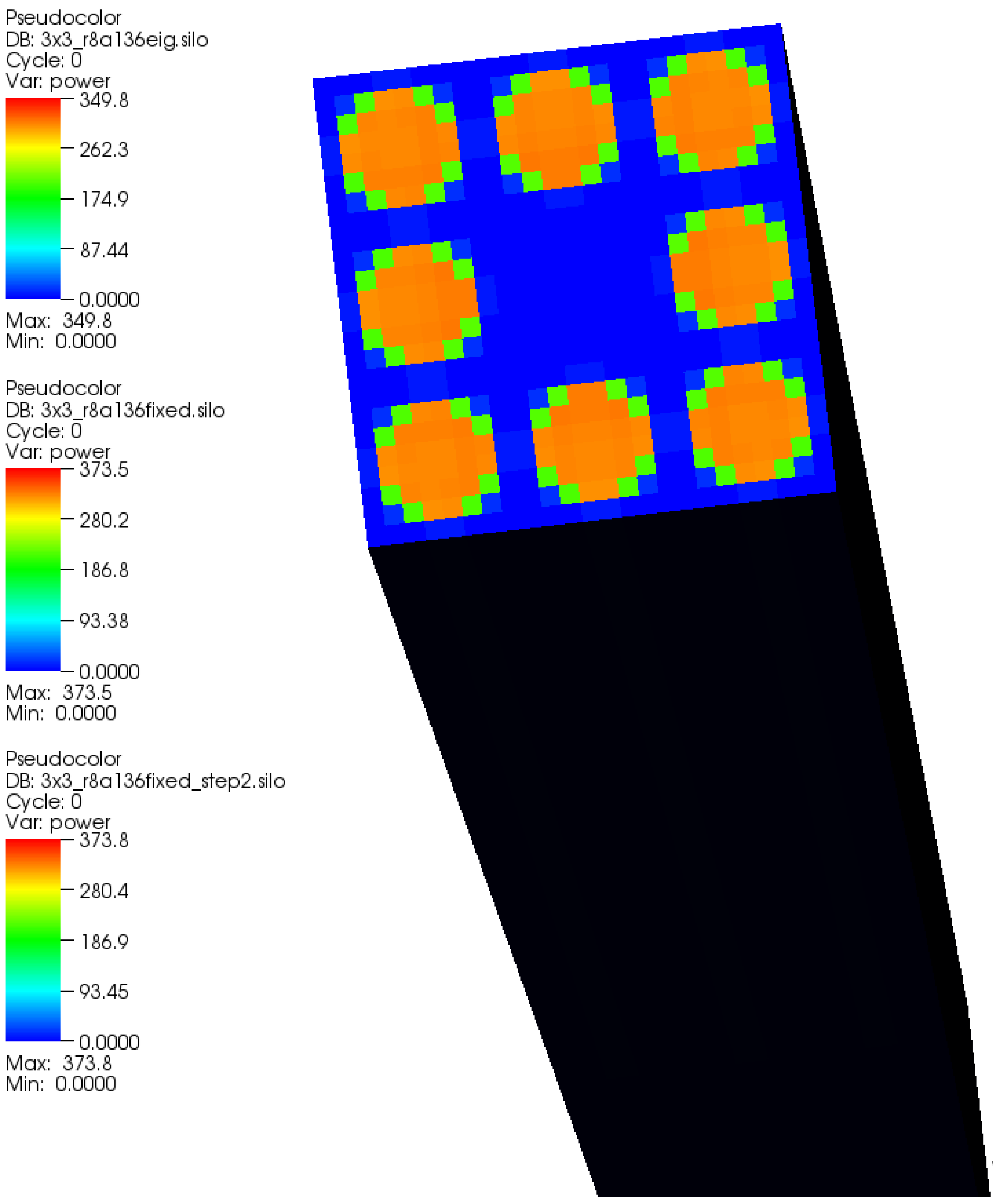

Figure 47 Eigenvalue, perturbed and analytic precursor source powers at 0.01 s

Chapter 7 CONCLUSIONS

7.1 SUMMARY

The objective of this dissertation was to develop a new kinetics method amenable to modern multi-physics and massively-parallel code suites that could be applied to solve the time-dependent Boltzmann transport equation with appropriate discretization strategies for time as well as space, angle, and energy. The semi-implicit direct kinetics (SIDK) method developed is demonstrated in both diffusion and transport formulations on fast and thermal reactors undergoing mild to severe transients for both FMFD and finite element method (FEM) spatial discretizations using from two to forty four neutron energy groups. The method is demonstrated to be as accurate to within ~0.2% on the same time mesh as direct kinetics, while executing an order of magnitude faster, for some problems, particularly those with larger numbers of neutron precursor groups, which constitutes a more accurate representation of delayed neutron generation rates. The SIDK method, for all problems considered, is shown to be within 0.2 % of direct kinetics and to execute at least twice as fast. The method is also demonstrated via a number of benchmark problems with comparisons to a number of time integration strategies, in which it performs well, with the exception of the discrepancies presented in the results for the LRA Benchmark, and which are primarily attributable to the difference in the approaches to thermal feedback treatment. The SIDK method is further demonstrated to be linearly convergent to exact analytical solutions. A new level of energy fidelity is demonstrated in a heterogeneous 3D control rod ejection problem run in parallel utilizing forty-four energy groups. A new level of within-pin thermal resolution is demonstrated via the two-way coupling of the AMP and Denovo codes.

In Chapter 1, the multi-scale nature of reactors and biases in current methods is explored. The chapter begins with motivation and organization as well as giving a brief description of computer codes used. This chapter included an overview of the steady-state eigenvalue and time-dependent Boltzmann transport equation as well as the isotopic depletion equations.

In Chapter 2, a literature review was presented with an overview of current multi-physics code efforts, core power distribution calculations for steady-state and time-dependent scenarios, as well as a detailed literature review of time-integration strategies. In this chapter, the multi-group diffusion equations were introduced as well as the time-dependent transport equation. The primary methods of time integration including implicit, explicit, and semi-implicit as well as multi-step and multi-stage methods were introduced. Also, the time integration schemes utilized for solutions of the point kinetics equations are herein presented, which include multi-stage Runge-Kutta methods, matrix exponential Pade approximant methods, and the semi-implicit Crank-Nicholson method.

In Chapter 3, the theory of the physics included in multi-scale reactor modeling is presented, and the accompanying motivation for the development of a robust large scale, transport based, computationally efficient neutron kinetics methodology. The new kinetics method, the semi-implicit direct kinetics (SIDK) method, is derived in reference to traditional kinetics methods, which include quasi-static, direct, and hybrid kinetics methods, in particular the HOBD method.

In Chapter 4, the Burner Reactor Integrated Safety Code (BRISC) is described and integrated with AMP. The integration of BRISC with AMP makes AMP the first fuel performance code to include an internal kinetics capability. An overview of the nuclear data generation process via the SCALE code suite is included. An overview of the standard methods applied to generate basic nuclear data for kinetics calculations (betas, lambdas, and reactivity coefficients) is provided. The BRISC package is verified externally and within AMP via the implementation of unit tests that check order of convergence, accuracy, as well as user options in a manner consistent with modern large-scale code development. BRISC's integration with AMP's thermal capabilities is explored, and a point kinetics model with thermal feedback capability is developed.

In Chapter 5, the SIDK method developed in Chapter 3 is implemented in stand-alone two-group time-dependent one and two-dimensional neutron diffusion implementations, to compare with neutron kinetics benchmarks as well as to explore multi-step, multi-stage, and adaptive time-integration strategies with and without thermal feedback. The accuracy is demonstrated to fall within 0.2 % of direct kinetics, while the speed is demonstrated to be an order of magnitude faster than direct kinetics. The accuracy, convergence, and speed of the new method are fully explored with regards to multi-step, multi-stage, and fully implicit solutions for fast and thermal systems with and without thermal feedback. The convergence is demonstrated to be limited to linear, even if higher order representations of power over the time step and temporal derivative of flux are used. The spatial convergence of the developed FMFD code is shown to be quadratic, as is the convergence of the multi-step, multi-stage fully-implicit TR-BDF2. The accuracy of the SIDK approach is demonstrated for short and longer times on fast and thermal reactors with and without feedback with regards to a number of other kinetics methods which feature alternative discretizations of space and time.

In Chapter 6, the necessary framework to implement the SIDK method in high fidelity three-dimensional parallel arbitrarily high number of energy group multi-physics approaches, developed in Chapter 3 and benchmarked in Chapter 5, is developed for the AMP Multiphysics framework, utilizing Denovo. The Denovo radiation transport code is an ideal candidate for the non-intrusive SIDK method, due to its massively parallel, verified, arbitrary multi-group, neutron transport, and modern nature, as well as being internally coupled within a multi-physics framework, which is AMP. The accuracy, convergence, and speed of the SIDK method are fully explored with regards to analytic point kinetics solutions as well as spatial kinetics transport benchmarks without feedback. The convergence is demonstrated to be linear to the exact solution. A true 3D control rod ejection study is then undertaken on a fully-resolved mini-assembly utilizing forty four energy groups.

The level of spatial and energy fidelity is higher than that employed by current kinetics methods. The control rod ejection problem compares favorably with point kinetics, and also provides the spatial power variation that is crucial in calculating local temperatures. The accuracy of this approach is demonstrated for short and longer times on representative fast and thermal systems without feedback with regards to a number of other kinetics methods which feature alternative discretizations of space, time, angle, and neutron energy, this approach being equal to the fidelity in space, with slightly lower fidelity in time and angle of the other approaches and superior in energy, thermal feedback resolution, and computational efficiency. Chapter 5 demonstrates ways to improve the temporal fidelity of the SIDK method, most applicable for problems without thermal feedback, while angular fidelity could come from saving the space-angular dependent fluxes, which adds a minimal amount of fidelity and a large memory or computational burden, particularly for multi-step methods (59) (14).

To summarize the SIDK method, its strength lies in determining power distributions with medium error tolerances, particularly on problems with large numbers of neutron precursor groups or thermal feedback where cross section update errors or higher order representation of power introduces more error than the first order convergence on the temporal derivative of the flux that the SIDK method is limited to. On problems where lower error tolerances are desired, higher order methods in time or a fine spatial mesh with the fully-implicit direct kinetics method is recommended. On problems with slow spatial shape change, quasi-static kinetics is recommended. Therefore, for high-fidelity massively parallel multi-physics simulation of severe transients with thermal-hydraulic feedbacks where medium error tolerances are needed on the temporal derivative of power and where employing multi-step methods would require saving multiple instances of the space-angle dependent flux, the SIDK approach is demonstrated as the best starting point due to its robust and computationally efficient nature.

7.2 CLOSING COMMENTS

It is demonstrated that for the fully-implicit direct kinetics, the first and second order SIDK, the second order HOBD, as well as the NDF solvers are all first order solution methods. Of all of the first order choices, the HOBD represents the best run time at coarse error tolerances on problems without thermal feedback, suffering no run time penalty versus the SIDK method and giving lower errors at very coarse time step sizes. The SIDK method represents the best choice at intermediate error tolerances, giving slightly lower errors at the same run times as the other methods, particularly on problems with thermal feedback. At low error tolerances, the TR-BDF2 solver is by far the most effective solver, yielding extremely low errors (1E-7 %) with fairly coarse time steps. However, the TR-BDF2 method is slow due to the multi-stage, multi-step process and is therefore not preferred at larger error tolerances. The TR-BDF2 method is demonstrated to be a second order method. Fully-implicit direct methods are preferred at fine space-time resolutions on severe problems with thermal feedback. As intermediate resolution and error tolerances make an excellent starting point for massively parallel calculations due to the necessity of not wasting computational time on debugging or overly coarse resolution, the SIDK method represents an excellent starting point for the rapid prototyping of large scale high-fidelity spatial kinetics methods.

A new implementation of a hybrid spatial kinetics method, the semi-implicit direct kinetics (SIDK) method, has been developed and applied within a multi-physics framework. As a result of this dissertation, Denovo can now solve transient reactor power distributions, in addition to the steady-state capabilities that already existed, which are three-dimensional eigenvalue and fixed source problems. The mini-assembly 3D control rod ejection run in parallel with Denovo represents a significant step forward in high-fidelity kinetics methods, particularly those coupled to multi-physics frameworks, in neutron energy as well as computational efficiency. The implementation of the SIDK method within the AMP-Denovo framework represents a capability that is more than legacy equivalent; through 3D space-time-dependent power distributions feeding 3D pin-resolved thermal responses, this capability is able to improve upon severe transient analysis.

Using these methods, the impact of highly resolved coupled power and temperature distributions during a control rod ejection transient were evaluated in contrast to current methodologies. Furthermore, this study demonstrated the fidelity and speed possible using the SIDK method, which is derived and benchmarked in this dissertation. This resolution, the speed of the method developed, and the highly amenable nature to modern multi-physics frameworks makes this an attractive option to developers of massively-parallel multi-physics codes looking to leverage steady-state radiation transport solvers. The SIDK method was developed and demonstrated to be accurate, fast, and significant with regard to eigenvalue and temperature prediction fidelity. In addition, viable pathways to possible future work and areas of improvement have been coherently and concisely presented.

WORKS CITED

1. **J. Duderstadt, L. Hamilton.** *Nuclear Reactor Analysis.* s.l. : John Wiley & Sons, Inc., 1976.

2. **IAEA.** *Current Trends in Nuclear Fuel for Power Reactors.* Vienna, Austria : IAEA, 2011.

3. **Wirth, Brian.** *Section 1: An Introduction to Materials Degradation in Nuclear Environments.* Knoxville, TN : University of Tennessee, January 2011.

4. **K. J. Geelhood, W. G. Luscher, C. E. Beyer, et al.** *Predictive Bias and Sensitivity in NRC Fuel Performance Codes.* Richland, WA : Pacific Northwest National Laboratory, October 2009. NUREG-CR-7001.

5. **Gary Berna, D.D Lanning.** *FRAPCON-3:Integral Assesment.* Richland, WA : s.n., 1997. NUREG/CR-6534, Vol. 3, PNNL-11513.

6. **Gehin, J. C.** *A Quasi-Static Polynomial Nodal Method for Nuclear Reactor Analysis.* Cambridge, MA : Massachusetts Institute of Technology, September 1992.

7. *A Unified Numerical Algorithm for Space-Dependent Kinetics Equations.* **T. Endo, Y. Ban, and A. Yamamoto.** 2005. Transactions of the American Nuclear Society. Vol. 104, pp. 865-867.

8. **T. Downar, et al.** *PARCS Theory Manual, U.S. NRC Core Simulator.* W. Lafayette, Indiana : Purdue University, 2004.

9. *Higher order backward discretization of the neutron diffusion equation.* **D. Ginestar, et al.** 1-3, s.l. : Annals of Nuclear Energy, 1998, Vol. 25. pg 47-64.

10. **K. Clarno, S. Hamilton, B. Philip, M. Berrill, R. Sampath, S. Allu, D. Pugmire, G. Dilts, J. E. Banfield.** *Integrated Radiation Transport and Nuclear Fuel Performance for Assembly-Level Simulations.* Oak Ridge, TN : s.n., 2012. ORNL/TM-2012/33.

11. *A New Semi-Implicit Direct Kinetics Method with Analytical Representation of Delayed Neutrons.* **J. E. Banfield, et al.** San Diego, CA : Transactions of the American Nuclear Society, 2012. Volume 107, pg 1111-1114.

12. **Society, Mathematics and Computation Division of the American Nuclear.** *National Energy Software Center: Benchmark Problem Book, Supplement 3.* Argonne, IL : Argonne National Laboratory, 1986. ANL-7416.

13. *A Time-Dependent, Three-Dimensional Neutron Transport Methodology.* **S. Goluoglu, H. L. Dodds.** s.l. : Nuclear Science and Engineering, 2001, Vol. 139, pp. 248-261.

14. *Higher Order Treatment on Temporal Derivative of Angular Flux for Time-Dependent MOC.* **Kosuke Tsujita, Tomohiro Endo, Akio Yamamoto, et al.** San Diego, CA : ANS TRANSACTIONS, 2012. Volume 107, pg 1101-1104.

15. *AMP - An Advanced Multi-Physics Fuel Performance Code.* **J. E. Banfield, K. Clarno, G. Ivan Maldonado.** North Myrtle Beach, SC : Validation, Verification and Uncertainty Quantification Conference, May 24-28, 2010. hosted by North Carolina State University.

16. *Quasi-Static Validation of the AMP Nuclear Fuel Performance Code.* **J. E. Banfield, et al.** Hollywood, FL : Trans. Am. Nucl. Soc., 2011. Vols. 104, 1, pp. 83-85.

17. *The AMP (Advanced MultiPhysics) Nuclear Fuel Performance Code.* **K. Clarno, B. Philip, W. Cochran, R. Sampath, S. Allu, P. Barai, S. Simunovic, M. Berrill, L. Ott, S. Pannala, G. Dilts, B. Mihaila, G. Yesilyurt, J. Lee, J. E. Banfield.** Oak Ridge, TN : s.n., 2012, Nuclear Engineering and Design, Vol. 252, pp. 108-120.

18. **K. Clarno, B. Philip. W. Cochran, G. Dilts, B. Mihaila, R. Sampath, S. Allu, P. Barai, G. Yesilyurt, J. E. Banfield, et al.** *User Manual for the AMP Nuclear Fuel Performance Code.* Oak Ridge, TN : Oak Ridge National Laboratory, September 2010.

19. **Sciences, National Institute of Computational.** 48.SpaceTime Neutron Kinetics within a MultiPhysics Framework for Nuclear Fuel Performance Applications. [Online] Kraken. [Cited: January January, 2013.] nics.tennessee.edu/science/current_projects. Project 48.

20. *Denovo - A New Three-Dimensional Parallel Discrete Ordinates Code in SCALE.* **T. Evans, et al.** s.l. : Nuclear Technology, 2010, Vol. 171, pp. 171-200.

21. *Solving Index-1 DAEs in MATLAB and Simulink.* **L. F. Shampine, M. W. Reichelt, J. A. Kierzenka.** s.l. : SIAM Review, 1999, Vol. 41, pp. 538-552.

22. **R. Schmidt, K. Belcourt, K. Clarno, R. Hooper, L. Humphries, A. Lorber, R. Pryor, W. Spotz.** *Foundational Development of an Advanced Nuclear Reactor Integrated Safety Code.* Albuquerque, NM : Sandia National Laboratories, February 2010. SAND2010-0878.

23. **The Math Works Inc.** *Getting Started Guid.* Natick, MA : s.n., 1984-2011.

24. **SCALE.** *SCALE: A Modular Code System for Performing Standardized Computer Analyses for Licensing Evaluation.* Available from Radiation Safety Information Computational Center. Oak Ridge, TN : Oak Ridge National Laboratory, 2005. Version 6.0. ORNL/TM-2005/39.

25. **Forrest Brown, Jeffrey Bull, John Goorley, Avneet Sood, Jeremy Sweezy.** *MCNP5-1.51 Release Notes.* Los Alamos, NM : Los Alamos National Laboratory, 2009.

26. **Agency, Nuclear Energy.** *Benchmark on Deterministic Transport Calculations Without Spatial Homogenization.* s.l. : Nuclear Science/NSC/DOC(2003) 16, 2003. ISBN 92-64-02139-6.

27. *Efficient numerical solutions of the point kinetics equations in nuclear reactor dynamics.* **M Kinard, E Allen.** s.l. : Annals of Nuclear Energy, 2004, Vol. 31, pp. 1039-1051.

28. **Wikipedia.** Wikipedia. [Online] [Cited: October 19, 2010.] http://en.wikipedia.org.

29. **Butcher, John C.** *Numerical methods for ordinary differential equations.* s.l. : John Wiley & Sons, 2003. ISBN 0471967580.

30. *The MATLAB ODE Suite.* **L. F. Shampine, M. W. Reichelt.** s.l. : SIAM Journal on Scientific Computing, 1997, Vol. 18, pp. 1-22.

31. **K. Clarno, J. E. Banfield.** *Mathematical formulation for the neutronics operators and their implementation in AMP.* Oak Ridge, TN : Oak Ridge National Laboratory, August 4, 2011. AMP Documentation.

32. **Hogle, Susan.** *Phd Dissertation.* The University of Tennessee-Knoxville : Nuclear Engineering, 2012.

33. *A spatial kinetic model for simulating VVER-1000 start-up transient.* **Samira Kashi, Nader Moghaddam, Shahriari Maleki.** s.l. : Annals of Nuclear Energy, 2011, Vol. 38, pp. 1300-1309.

34. **K. J. Geelhood, W. G. Luscher, C. E. Beyer, et al.** *FRAPTRAN 1.4: A Computer Code for the Transient Analysis of Oxide Fuel Rods.* Richland, WA : Pacific Northwest National Laboratory, March 2011.

35. **Yagnik, S.** *Fuel Analysis and Licensing Code: FALCON MOD01: Volume 2: User's Manual.* Palo Alto, CA : s.n., 2004. 1011308.

36. *PWR Contorl Rod Ejection Analysis with the MOC Code DECART.* **M. Hursin, T. Downar.** Berkley, CA : Joint International Workshop: Nuclear Technology Society-Needs for Next Generation, January 6-8, 2008.

37. *Coupled Radiation Transport and Thermomechanics using the AMP and Denovo Codes.* **S. Hamilton, K. Clarno, B. Philip, R. Sampath, M. Berrill, M. Baird, J. E. Banfield.** Copper Mountain, CO : Twelfth Copper Mountain Conference on Iterative Methods, March 25-30, 2012.

38. **Commission, Nuclear Regulatory.** NRC. *10 CFR 50.36 Technical Specification.* [Online] http://www.nrc.gov.

39. **Doug Kothe, Ronaldo Szilard, Paul Turinsky.** *CASL: The Consortium for Advanced Simulation of Light Water Reactors.* Oak Ridge, TN : US Department of Energy, 2010.

40. NEAMS. *Nuclear Energy Advanced Modeling and Simulation Program.* [Online] [Cited: December 17, 2012.] neams.ne.anl.gov.

41. F-Bridge. *F-BRIDGE PROJECT.* [Online] European Union. [Cited: December 17, 2012.] www.f-bridge.eu.

42. *Three dimensional coupled simulation of thermomechanics, heat, and oxygen diffusion in UO2 nuclear fuel rods.* **C. Newman, G. Hansen, D. Gaston.** 1, s.l. : Journal of Nuclear Materials, July 2009, Vol. 392, pp. 6-15.

43. *ALCYONE: the Pleiades fuel performance code dedicated to multidimensional PWR studies.* **G. Thouyenin, J. M. Ricaud, D. Planca, P. Thevenin.** Salamanaca, Spain : In proceedings of Top Fuel 2006, October 2006.

44. Consortium for Advanced Simulation of LWRs. *CASL.* [Online] Department of Energy. [Cited: December 17, 2012.] www.casl.gov/highlights/radiation_transport.shtml.

45. *Transient Capability for a MOC-Based Whole Core Transport Code DeCART.* **Jin-Young Cho, et al.** s.l. : Trans. Am. Nucl. Soc., 2005. Vol. 92, pp. 721-722.

46. **Texas, University of.** Source Forge. [Online] [Cited: July 12, 2011.] http://libmesh.sourceforge.net.

47. **C. Rabiti, M. A. Smith, W. Yang, G. Palmiotti, et al.** *Status Report on SHARP Coupling Framework.* s.l. : Argonne National Laboratory, 2007.

48. *Stability of the SUPG finite element method for transient advection-diffusion problems.* **Pavel B. Bochev, Max D. Gunzburger, John N. Shadid.** 193, s.l. : Computer methods in applied mechanics and engineering, 2004, pp. 2301-2323.

49. **R. Courant, K. Friedrichs, H. Lewy.** *On the partial difference equations of mathematical physics.* New York, NY : AEC Research and Development Report - Courant Institute of Mathematical Sciences, September 1956 [1928]. p. V +76. This is an earlier version of the paper Courant, Friedrichs & Lewy circulated as a research report. NYO-7689.

50. **Andersson, Christer.** *Time-stepping schemes for phase-field simulation of dendritic solidifcation.* Royal Institute of Technology. Stockholm, Sweeden : Department of Numerical Analysis and Computer Science, July 2002. TRITA-NA-0216.

51. **A Schubert, C. Gyori, D. Elenkov, K. Lassmann, J. Van de Laar.** *Analysis of Fuel Centre Temperatures with the TRANSURANUS Code.* Wurzburg, Germany : Bulgarian Academy of Sciences, March 2003.

52. **C.M Allison, G. A. Berna, R. Chambers, et al.** *SCDAP/RELAP5/MOD3.1 Code Manual.* Idaho National Engineering Laboratory. Idaho Falls, Idaho : s.n., November, 1993. Volume IV: MATPRO - A Library of Materials Properties for Light-Waater-Reactor-Accident Analysis.

53. *Thermal-hydraulic behaviour of a marine reactor during oscillations.* **I. Ishihida, T. Kusunoki, H. Murata, T. Yokomura, M. Kobayashi, et al.** 2-3, s.l. : Nuclear Engineering and Design, June 1990, Vol. 120, pp. 213-225.

54. **NESTLE.** *NESTLE Version 5.2.1.* Raleigh, NC : NC State University, July 2003.

55. *Black Rabbit Ejection Studies and COMSOL Kinetics Modeling Development at HFIR.* **D. Chandler, G. I. Maldonado, T. R. Primm, R. Hobbs.** Hollywood, FL : Trans. Am. Nucl. Soc., June 2011.

56. **Clarno, K.** *Rascal 2-D Multi-Group Reactor Diffusion Code.* Decmber 15, 2008.

57. **J. M Aragones, C. Diop, U. Rohde.** *Nuclear Reactor Integrated Simulation Project.* s.l. : NURISP - Document D-1.0, 3/22/2010.

58. **Stacey, M. Weston.** *Nuclear Reactor Physics.* Weinheim : WILEY-VCH Verlag GmbH & Co. KGaA, 2007.

59. **A. Hoffman, et al.** *Transient Methods for Neutron Transport.* s.l. : CASL RTM PI Meeting, 2012.

60. **Canteach.** [Online] [Cited: December 09, 2010.] http://canteach.candu.org.

61. **Project, OECD Halden Reactor.** Halden Boiling Water Reactor. [Online] Institutt for energiteknikk, January 2003. www.ife.no.

62. **Team, RELAP5.** *RELAP5-3D Code Manual Volume 1: Code Structure, System Models, and Solution Methods.* Idaho Falls, ID : Idaho National Laboratory, April 2005.

63. **H.L Dodds, R. M. Westfall.** *SKINATH - A Computer Program for Solving the Reactor Point Kinetics Equations with Simple Thermal-Hydrualic Feedback.* Oak Ridge, TN : Oak Ridge National Laboratory, 1984. ORNL/CSD/TM-210.

64. **Laboratory, Lawrence Livermore National.** SUNDIALS. [Online] [Cited: April 10, 2010.] https://computation.llnl.gov/casc/sundials/main.html.

65. **Laboratory, Sandia National.** Trilinos. [Online] [Cited: April 15, 2010.] http://trilinos.sandia.gov/.

66. *Jacobian-free Newton-Krylov methods: A survey of approaches and applications.* **D. A. Knoll, D. E Keyes.** s.l. : Journal of Computational Physics, 2004, Vol. 193, pp. 357-397.

67. **GNU.** gnu.org. *GNU Scientific Library.* [Online] Open Source.

68. *Benchmarking of Software and Methods for use in Transient Multidimensional Fuel Performance with Spatial Reactor Kinetics.* **J. E. Banfield, et al.** Chicago, IL : Proceedings of International Conference on Advances in reactor Power Physics, 2012.

69. *A Validation Study of Pin Heat Transfer for UO2 Fuel Based on the IFA-432 Experiments.* **A. Phillippe, K. Clarno, J. Banfield, et al.** s.l. : Nuclear Science and Engineering, 2012, Vol. (in submission process).

70. *A Validation Study of Pin Heat Transfer for MOX Fuel Based on the IFA-597 Experiments.* **A. Phillippe, K. Clarno, J. Banfield, et al.** s.l. : Nuclear Science and Engineering, 2012, Vol. (in submission process).

71. **OECD, Secretary-General of the.** *Nuclear Fuel Behavior Under Reactivity-initiated Accidents (RIA) Conditions.* s.l. : Nuclear Energy Agency, 2010. nea6847-behaviour-RIA.

72. **D. D. Lanning, C. E. Beyer, K. J. Geelhood.** *FRAPCON-3 Updates, Including Mixed-Oxide Fuel Properties.* Washington DC : U.S. Nuclear Regulatory Commission Office of Nuclear Regulatory Research, May 2005. Volume 4. NUREG/CR-6534, PNNL-11513.

73. **D. D. Lanning, C. E. Beyer, C. L. Painter.** *FRAPCON-3: Modifications to Fuel Rod Material Properties and Performance Models for High-Burnup Application.* Richland, WA : Pacific Northwest National Laboratory, October 1997. NUREG/CR-6524 Volume 1, PNNL-11513.

74. **Hart, Shane.** NE Cluster Wiki. [Online] University of Tennessee. necluster.engr.utk.edu/wiki.

75. *Effect of Fuel Temperature Profile on Eigenvalue Calculations.* **T. Greifenkamp, K. T. Clarno, J. C. Gehin.** s.l. : American Nuclear Society, Student Conference, 2008.

76. **Jesse, Matt.** R&D Staff Member in Reactor and Nuclear Systems Division at ORNL. *Personal Communication.* Oak Ridge, TN, March 18, 2013.

77. **SCALE.** *A Comprehensive Modeling and SImulation Suite for Nuclear Safety Analysis and Design.* Oak Ridge, TN : Available from Radiation Safety Information Computational Center at Oak Ridge National Laboratory as CCC-785, June, 2011. Version 6.1 ORNL/TM-2005/39.

APPENDIX A

TRITON input

```
'Input generated by GeeWiz SCALE 6.0.2 Compiled on February 18, 2009
=t-newt parm=(centrm)
HALDEN pin cell
v7-238
read composition
 uo2         1 0.955 607
                                 92234 0.005407837
                                 92235 10
                                 92238 89.99459   end
 helium      2 1 525   end
 zirc4       3 1 520   end
 d2o         4 1 508   end
end composition
read celldata
  latticecell squarepitch fuelr=0.534035 1 gapr=0.545465 2 cladr=0.639445 3
hpitch=0.7112 4 end
end celldata
READ KEEP_OUTPUT
  newt
END KEEP_OUTPUT
read model
  READ PARAMETER
    drawit=yes  run=yes epsouter=-1e-2  epseigen=1e-8 epsinner=1e-9 timed=yes
    sn=10   inners=100 outers=150  echo=yes prtmxsec=no prtmxtab=no
    cmfd=yes xycmfd=1 converg=mix collapse=yes  prtxsec=no
    prtbroad=no prthmmix=yes
  END PARAMETER
READ MATERIALS
 1 1 !fuel   ! end
 2 1 !helium ! end
 3 1 !clad   ! end
 4 1 !coolant! end
END MATERIALS
READ COLLAPSE
  200r1 38r2
END COLLAPSE
READ HMOG
101 pincell 1 end
102 pincell 2 end
103 pincell 3 end
104 pincell 4 end
END HMOG
read geometry
global unit 1
 cylinder 1  0.534035
 cylinder 2  0.545465
 cylinder 3  0.639445
 cuboid 4  0.7112  -0.7112 0.7112 -0.7112
 media 1 1 1
 media 2 1 2 -1
 media 3 1 3 -2
 media 4 1 4 -3
 boundary 4 4 4
end geometry
read bounds
    all=refl
  end bounds
end model
end
```

APPENDIX B

CSAS Input Example

```
'Input generated by GeeWiz SCALE 6.1 Compiled on Mon Jun  6 11:04:33 2011
'batch_args \-x\-m
=csasi
csas-halden
v7-238
read composition
 uo2         1 0.955 300
                                 92234 0.005407837
                                 92235 10
                                 92238 89.99459   end
 helium      2 1 300   end
 zirc4       3 1 300   end
 d2o         4 1 300   end
 b4c         5 1 300   end
end composition
read celldata
  latticecell squarepitch fuelr=0.534035 1 gapr=0.545465 2 cladr=0.639445 3
hpitch=0.7112 4 end
end celldata
end
=shell
   cp ft02f001 "$HOME/new_SCALE/Halden-CSAS_init.lib"
end
```

DENOVO Eigenvalue – Halden Case

```
##############################################################################
##
## Denovo HALDEN 3X3
##
##############################################################################
## Copyright (C) 2008 Oak Ridge National Laboratory, UT-Battelle, LLC.
##--------------------------------------------------------------------------##
## generated by /data/denovo/build/debug/bin/pygen built on 20101129
##############################################################################

import os, sys, math, string

# pykba equation type
from sc import *
unks_cell = 1

Npin = 8 # number of cells per pin in coarse mesh
Ng = 44   #number of energy groups on the AMPX library
Zmax = 410.
pn_order = 0

store_bound = 1

run_problem     = 1
generate_hpckba = 0

# Starting z value (this places bottom of fuel at z=0)
Zmin         = 0.0
```

```
# Number of different axial material levels
num_mat_levels = 3

# Pin spacing
pitch = 1.4224

# Clad inner/outer radius
clad_in  = 0.545465
clad_out = 0.639445

# Guid tube inner/outer radius
gt_in  = 0.569
gt_out = 0.6448

lat_size = 3
Nx = lat_size*Npin
Ny = lat_size*Npin
plane_cells = Nx*Ny

store_bound = 0

run_problem     = 1
generate_hpckba = 0


# Parameters for region 0
fuel_id0 = 101
clad_id0 = 102
mod_id0  = 103
axial_cells_region0 = 64
delta_z_region0 = 5.59375

# Parameters for region 1
fuel_id1 = 201
clad_id1 = 202
mod_id1  = 203
cr_id1   = 204
axial_cells_region1 = 24
delta_z_region1 =  0.015625

# Parameters for region 2
fuel_id2 = 301
clad_id2 = 302
mod_id2  = 303
cr_id2   = 304
axial_cells_region2 = 48
delta_z_region2 =  0.8671875

Nz = axial_cells_region0 + axial_cells_region1 + axial_cells_region2

##--------------------------------------------------------------------------##
## BUILD MESH
##--------------------------------------------------------------------------##
print "Defining build_mesh"

def build_mesh(N):
    # uniform layout for reflector/plate regions
    uniform_layout = [1, 1, 1,
                      1, 1, 1,
                      1, 1, 1]

    # fuel lattice arrangement
    fuel_layout = [1, 1, 1,
                   1, 2, 1,
                   1, 1, 1]

    # Bottom reflector
    lattice = []
    for k in xrange(num_mat_levels):
        lattice[k:] = [Array_0(lat_size)]

    # Number of clean materials per level
    num_mat = [0] * num_mat_levels

    # Number of clean materials total
    num_clean_mat = 0

    # Bottom region
    print "Building level  0"
    pin = Pincell()
    pin.set_pitch( pitch )
    r = [clad_in, clad_out]
    ids = [ fuel_id0, clad_id0 ]
    pin.set_shells(ids, r, mod_id0)

    gt = Pincell()
    gt.set_pitch( pitch )
    r = [gt_in, gt_out]
    ids = [mod_id0, clad_id0]
    gt.set_shells(ids, r, mod_id0)

    # Assign reflector pin to bottom level of lattice
    lattice[0].set_objects(fuel_layout)
    lattice[0].assign_object(pin,1)
    lattice[0].assign_object(gt,2)
    lattice[0].build_array(N,0)

    # Set global mixing table
    num_mat[0] = lattice[0].num_mat()
    mix_table = lattice[0].mixing_vector()
    num_clean_mat = num_mat[0]

    # Create all other levels
    for k in xrange(1,num_mat_levels):
        print "Building level ",k

        # Fuel regions
        if k==1:
            pin = Pincell()
            pin.set_pitch( pitch )
            r = [clad_in, clad_out]
            ids = [fuel_id1, clad_id1]
            #ids = [cr_id1, clad_id1]
            pin.set_shells(ids, r, mod_id1)

            gt = Pincell()
            gt.set_pitch( pitch )
            r = [gt_in, gt_out]
            #ids = [fuel_id1, clad_id1]
            ids = [cr_id1, clad_id1]
            gt.set_shells(ids, r, mod_id1)

            # Build lattice for this level
            lattice[k].set_objects(fuel_layout)
            lattice[k].assign_object(pin,1)
            lattice[k].assign_object(gt,2)

        # Fuel regions
        if k==2:
            pin = Pincell()
            pin.set_pitch( pitch )
            r = [clad_in, clad_out]
            ids = [fuel_id2, clad_id2]
            #ids = [cr_id1, clad_id1]
            pin.set_shells(ids, r, mod_id2)

            gt = Pincell()
            gt.set_pitch( pitch )
            r = [gt_in, gt_out]
            #ids = [fuel_id1, clad_id1]
            ids = [cr_id2, clad_id2]
            gt.set_shells(ids, r, mod_id2)

            # Build lattice for this level
            lattice[k].set_objects(fuel_layout)
            lattice[k].assign_object(pin,1)
            lattice[k].assign_object(gt,2)

        # Finish lattice construction for this level
        print "Building array"
        lattice[k].build_array(N,N,0,mix_table,num_clean_mat)
        print "Done building array"

        # Update global mixing table
        num_mat[k] = lattice[k].num_mat()
        mix_table  = lattice[k].mixing_vector()

        if num_mat[k]>num_clean_mat:
            num_clean_mat = num_mat[k]

        # Update lower level matids
        print "Updating old mat ids"
        for j in xrange(k):
            lattice[j].set_mixids( lattice[k].update_old_matids( \
                lattice[j].num_mat(), lattice[j].mixids() ) )
            lattice[j].set_num_mat( num_clean_mat )

    # Build axial mesh
    z = [0.0] * (Nz + 1)
    z[0] = Zmin
    for k in xrange(Nz):
        if k < axial_cells_region1:
            z[k+1] = z[k] + delta_z_region0
        if (k >=axial_cells_region1 and k < axial_cells_region2):
            z[k+1] = z[k] + delta_z_region1
        if (k>=axial_cells_region2):
            z[k+1] = z[k] + delta_z_region2
    # mesh planes in x,y
    xy  = lattice[0].xy_planes()
```

```
    # Set up mixture ids for all cells
    mixids = Vec_Int(Nz*Ny*Nx, 0)

    # Lower material region
    offset = 0
    for local_k in xrange(axial_cells_region0):
        k = local_k + offset
        xyids = lattice[0].mixids()
        for j in xrange(Ny):
            for i in xrange(Nx):
                cell = i + Nx * (j + k*Ny)
                plane_cell = i + Nx*j
                mixids[cell] = xyids[plane_cell]

    # Upper material region
    offset = axial_cells_region0
    for local_k in xrange(axial_cells_region1):
        k = local_k + offset
        xyids = lattice[1].mixids()
        for j in xrange(Ny):
            for i in xrange(Nx):
                cell = i + Nx * (j + k*Ny)
                plane_cell = i + Nx*j
                mixids[cell] = xyids[plane_cell]

    # Upper material region - Control rod In
    offset = axial_cells_region0 + axial_cells_region1
    for local_k in xrange(axial_cells_region2):
        k = local_k + offset
        xyids = lattice[2].mixids()
        for j in xrange(Ny):
            for i in xrange(Nx):
                cell = i + Nx * (j + k*Ny)
                plane_cell = i + Nx*j
                mixids[cell] = xyids[plane_cell]

    # list of ids
    ids    = Vec_Int(num_clean_mat)
    for m in xrange(len(ids)):
        ids[m] = m

    return (xy, z, mixids, ids, mix_table, lattice)

##---------------------------------------------------------------------------##
## MAIN
##---------------------------------------------------------------------------##
print 'Initializing'
initialize(sys.argv)

if node() == 0:
    print "Denovo - pykba Python Front-End"
    print "-------------------------------"
    print "Release      : %16s" % (release())
    print "Release Date : %16s" % (release_date())
    print "Build Date   : %16s" % (build_date())
    print

timer = Timer()
timer.start()
```

```
##---------------------------------------------------------------------------##
## DB
##---------------------------------------------------------------------------##

print 'Building database'
db = DB("pykba")

# problem type
db.insert("problem_type", "EIGENVALUE")

db.insert("num_z_blocks", 1)
# decomposition
if nodes() == 1:

    db.insert("num_blocks_i", 1)
    db.insert("num_blocks_j", 1)
    iblocks = 1
    jblocks = 1

elif nodes() == 2:

    db.insert("num_blocks_i", 2)
    db.insert("num_blocks_j", 1)
    iblocks = 2
    jblocks = 1

elif nodes() == 4:

    db.insert("num_blocks_i", 2)
    db.insert("num_blocks_j", 2)
    iblocks = 2
    jblocks = 2

elif nodes() == 8:

    db.insert("num_blocks_i", 4)
    db.insert("num_blocks_j", 2)
    iblocks = 4
    jblocks = 2

elif nodes() == 16:

    db.insert("num_blocks_i", 4)
    db.insert("num_blocks_j", 4)

elif nodes() == 32:

    db.insert("num_blocks_i", 8)
    db.insert("num_blocks_j", 4)

elif nodes() == 64:

    db.insert("num_blocks_i", 8)
    db.insert("num_blocks_j", 8)

elif nodes() == 96:

    db.insert("num_blocks_i", 12)
    db.insert("num_blocks_j", 8)
```

```
#db.insert("num_blocks_i", 1)
#db.insert("num_blocks_j", 1)
db.insert("num_sets",     1)

# energy partitioning
db.insert("partition_upscatter", 1, 1)

# data settings
db.insert("num_groups", Ng)
db.insert("downscatter", 0, 1)
db.insert("Pn_order", pn_order)

# solver setup
db.insert("eigen_solver", "arnoldi")
db.insert("mg_solver", "krylov")
#db.insert("within_group_solver","GMRES_R")
db.insert("tolerance", 1.0e-5)
db.insert("aztec_kspace", 30)
db.insert("max_itr", 50)

db.insert("iterate_downscatter", store_bound, 1)
db.insert("use_init_guess", store_bound, 1)

# eigenvalue information
db.add_db("eigenvalue_db", "eigenvalue")
db.insert("eigenvalue_db", "L2_tolerance", 1e-5)
#db.insert("eigenvalue_db", "k_tolerance", 1e-5)
db.insert("eigenvalue_db", "diagnostic_level", 2)
db.insert("eigenvalue_db", "keff", 1.0)
db.insert("eigenvalue_db", "inner_tol_relaxer", "CONSTANT")
db.insert("eigenvalue_db", "inner_tol_relax_factor",1.0)
db.insert("eigenvalue_db", "arnoldi_restarts", 10)
db.insert("eigenvalue_db", "arnoldi_kspace", 15)
db.insert("eigenvalue_db", "energy_dep_ev", 1, 1)
db.insert("eigenvalue_db", "calculate_moments", 1, 1)

# upscatter database
db.add_db("upscatter_db", "upscatter")
db.insert("upscatter_db", "tolerance", 1.0e-6)
db.insert("upscatter_db", "aztec_diag", 0)
db.insert("upscatter_db", "aztec_output", 0)

# Mesh
(x, z, matids, cleanids, table, lattice) = build_mesh(Npin)
print "Done with build_mesh()"
db.insert("x_edges", x)
db.insert("y_edges", x)
db.insert("z_edges", z)

## Boundary conditions
bounds = [1, 1, 1, 1, 0, 0]
db.insert("boundary", "reflect")
db.add_db("boundary_db", "bnd_conditions")
db.insert("boundary_db", "reflect", bounds, 1)
db.insert("boundary_db", "store_bnd_state", store_bound, 1)

# Angular options
db.add_db("quadrature_db", "quad_options")
```

```
#db.insert("quadrature_db", "quad_type", "qr")
#db.insert("quadrature_db", "quad_type", "ldfe")
#db.insert("quadrature_db", "order", 3)
db.insert("quadrature_db", "Sn_order", 6)
#db.insert("quadrature_db", "quad_type", "glproduct")
#db.insert("quadrature_db", "polars_octant", 2)
#db.insert("quadrature_db", "azimuthals_octant",2)

##---------------------------------------------------------------------------##
## MANAGER
##---------------------------------------------------------------------------##

# make manager, material, and angles
manager = Manager()
mat     = Mat()
angles  = Angles()

# partition the problem
print "Manager partitioning"
manager.partition(db, mat, angles)
print "Manager done partitioning"

# get mapping and mesh objects
mapp    = manager.get_map()
indexer = manager.get_indexer()
mesh    = manager.get_mesh()

# global and local cell numbers
Gx = indexer.num_global(X)
Gy = indexer.num_global(Y)
Gz = mesh.num_cells_dim(Z)
Nx = mesh.num_cells_dim(X)
Ny = mesh.num_cells_dim(Y)
Nz = mesh.num_cells_dim(Z)

if node() == 0:
    print ">>> Partitioned global mesh with %i x %i x %i cells" \
          % (Gx, Gy, Gz)

##---------------------------------------------------------------------------##
## MATERIAL SETUP
##---------------------------------------------------------------------------##

# AMPX library
ampx = AMPX()
ampx.read_AMPX("Halden-CSAS.lib")

xsdb = XS_DB(db)
xsdb.set_num(405)
xsdb.assign_ampx(fuel_id0, 1, ampx)
xsdb.assign_ampx(fuel_id1, 1, ampx)
xsdb.assign_ampx(fuel_id2, 1, ampx)
xsdb.assign_ampx(clad_id0, 2, ampx)
xsdb.assign_ampx(clad_id1, 2, ampx)
xsdb.assign_ampx(clad_id2, 2, ampx)
xsdb.assign_ampx(mod_id0,  4, ampx)
xsdb.assign_ampx(mod_id1,  4, ampx)
xsdb.assign_ampx(mod_id2,  4, ampx)
xsdb.assign_ampx(cr_id1,   5, ampx)
xsdb.assign_ampx(cr_id2,   5, ampx)
```

```
beta_t = .007249
steady_precursors=Vec_Dbl(mesh.num_cells(),0.0)

##---------------------------------------------------------------------------##
## DELAYED GROUPS
##---------------------------------------------------------------------------##

beta=Vec_Dbl(6,0.0)

beta[0]=2.43e-04
beta[1]=1.45E-03
beta[2]=1.34e-03
beta[3]=2.92e-03
beta[4]=1.04e-03
beta[5]=2.56e-04

lamb=Vec_Dbl(6,0.0)

lamb[0]= 1.27e-02
lamb[1]= 3.17e-02
lamb[2]= 1.17e-01
lamb[3]= 3.14e-01
lamb[4]= 1.38
lamb[5]= 3.83

# make macro mixer
mixer = Macro_Mixer(xsdb)
mixer.set(cleanids, table)

# make the material database
mixer.mix_with_global_ids(matids, mat)

##---------------------------------------------------------------------------##
## ENERGY PARTITIONING
##---------------------------------------------------------------------------##

print "Partitioning energy"
manager.partition_energy(mat, angles)
erg_set = manager.get_erg_set_comm()

##---------------------------------------------------------------------------##
## SOURCE SETUP
##---------------------------------------------------------------------------##

# allocate problem state (use a zero source)
print "Allocating state"
source = Zero_Source()
manager.setup(source)

##---------------------------------------------------------------------------##
## SOLVE
##---------------------------------------------------------------------------##

if node() == 0:
    print ">>> Setup complete"
    print ">>> Solving with %s differencing and %s quadrature (%i angles)" \
          % (manager.spatial_descriptor(), angles.quad_label(), angles.num_angles())

if run_problem==1:
```

```
print "Running problem"

# solve the problem
manager.solve(angles)

steady_precursors= Vec_Dbl(mesh.num_cells(),0.0)


power = Vec_Dbl(mesh.num_cells(),0.0)
for cell in xrange(mesh.num_cells()):
    matid = mat.matid(cell)
    if mat.assigned_fission(matid):
        for g in xrange(Ng):
            phi = Moments(g)
            power[cell] +=  phi.scalar_flux(cell) \
                    * mat.fission_data(matid, g, NU_SIGMA_F)

for cell in xrange(mesh.num_cells()):
    for i in xrange(6):
        steady_precursors[cell]= steady_precursors[cell] + beta[i]*power[cell]


filename= open('precursors'+str(node())+'.txt','w')
filename.write(str(mesh.num_cells())+'\n')
for cell in xrange(mesh.num_cells()):
    filename.write(str(mapp.l2g(cell))+' '+str(steady_precursors[cell])+'\n')

filename.close()

flux0 = Vec_Dbl(mesh.num_cells(), 0.0)
flux5 = Vec_Dbl(mesh.num_cells(), 0.0)
flux10 = Vec_Dbl(mesh.num_cells(), 0.0)
flux15 = Vec_Dbl(mesh.num_cells(), 0.0)
flux20 = Vec_Dbl(mesh.num_cells(), 0.0)
flux25 = Vec_Dbl(mesh.num_cells(), 0.0)
flux30 = Vec_Dbl(mesh.num_cells(), 0.0)
flux35 = Vec_Dbl(mesh.num_cells(), 0.0)
flux40 = Vec_Dbl(mesh.num_cells(), 0.0)

phi = Moments(0)
for cell in xrange(mesh.num_cells()):
    flux0[cell]  = phi.scalar_flux(cell)

phi = Moments(5)
for cell in xrange(mesh.num_cells()):
    flux5[cell]  = phi.scalar_flux(cell)

phi = Moments(10)
for cell in xrange(mesh.num_cells()):
    flux10[cell] = phi.scalar_flux(cell)

phi = Moments(15)
for cell in xrange(mesh.num_cells()):
    flux15[cell] = phi.scalar_flux(cell)

phi = Moments(20)
for cell in xrange(mesh.num_cells()):
    flux20[cell] = phi.scalar_flux(cell)
    phi = Moments(25)
    for cell in xrange(mesh.num_cells()):
        flux25[cell] = phi.scalar_flux(cell)

    phi = Moments(30)
    for cell in xrange(mesh.num_cells()):
        flux30[cell] = phi.scalar_flux(cell)

    phi = Moments(35)
    for cell in xrange(mesh.num_cells()):
        flux35[cell] = phi.scalar_flux(cell)

    phi = Moments(40)
    for cell in xrange(mesh.num_cells()):
        flux40[cell] = phi.scalar_flux(cell)


    silo = SILO()
    silo.add_mixer(mixer)

    silo.open(str(lat_size)+"x"+str(lat_size)+"_r"+str(Npin)+"a"+str(Nz)+'eig')

    silo.add("flux0",  flux0)
    silo.add("flux5",  flux5)
    silo.add("flux10", flux10)
    silo.add("flux15", flux15)
    silo.add("flux20", flux20)
    silo.add("flux25", flux25)
    silo.add("flux30", flux30)
    silo.add("flux35", flux35)
    silo.add("flux40", flux40)
    silo.add("power",  power)

    silo.close()

# Write HPCKBA input
if generate_hpckba==1:

    print "Generating HPCKBA input"

    out = HPC_Problem_Output(1, Nx, Ny, Nz)

    print out.chunk(), out.num_chunks_per_file(), out.num_files()

    out.open(str(lat_size)+"x"+str(lat_size)+"_r"+str(Npin)+"a"+str(Nz))

    shapes = Vec_Dbl()

    out.write_db(db)
    out.write_mixed_xs(mixer)
    out.write_src_info(ZERO_SOURCE, ZERO_SOURCE, shapes)

    # write matids
    out.start_field_loop()
    ids = Vec_Int(out.chunk(), 0)
    while not out.finished_field_loop():
        k   = out.current_chunk()
        if k < Nz:
           print "Writing k-plane %d/%d" % (k, Nz)
           for j in xrange(Gy):
                for i in xrange(Gx):
                    cell  = indexer.g2g(i,j,k)
                    index = indexer.g2g(i,j,0)
                    ids[index] = matids[cell]
        out.write_matids(ids)
        out.advance_loop()

    out.close()
##---------------------------------------------------------------------------##
## TIMING
##---------------------------------------------------------------------------##

# output final database (has class-dependent defaults)
db.output()

timer.stop()
time = timer.wall_clock()

keys = timer_keys()
if len(keys) > 0 and node() == 0:
    print "\n"
    print "TIMING : Problem ran in %16.6e seconds." % (time)
    print "------------------------------------------------"
    for key in keys:
        print "%30s : %16.6e" % (key, timer_value(key) / time)
    print "------------------------------------------------"

##---------------------------------------------------------------------------##

manager.close()
finalize()

###############################################################################
## end
###############################################################################
```

DENOVO Fixed Source Step 1 – Halden Case

```
###############################################################################
##
## Denovo HALDEN 3X3 - Step 1, CRout
##
###############################################################################
## Copyright (C) 2008 Oak Ridge National Laboratory, UT-Battelle, LLC.
##---------------------------------------------------------------------------##
## generated by /data/denovo/build/debug/bin/pygen built on 20101129
###############################################################################

import os, sys, math, string

# pykba equation type
from sc import *
unks_cell = 1

Npin = 8 # number of cells per pin in coarse mesh
Ng = 44   #number of energy groups on the AMPX library
Nz = 40
Zmax = 410.
```

```
pn_order = 0

store_bound = 1

run_problem     = 1
generate_hpckba = 0

# Starting z value (this places bottom of fuel at z=0)
Zmin         = 0.0

# Number of different axial material levels
num_mat_levels = 3

# Pin spacing
pitch = 1.4224

# Clad inner/outer radius
clad_in  = 0.545465
clad_out = 0.639445

# Guid tube inner/outer radius
gt_in  = 0.569
gt_out = 0.6448

lat_size = 3
Nx = lat_size*Npin
Ny = lat_size*Npin
plane_cells = Nx*Ny

store_bound = 0

run_problem     = 1
generate_hpckba = 0

# Parameters for region 0
fuel_id0 = 101
clad_id0 = 102
mod_id0  = 103
axial_cells_region0 = 64
delta_z_region0 = 5.59375

# Parameters for region 1
fuel_id1 = 201
clad_id1 = 202
mod_id1  = 203
cr_id1   = 204
axial_cells_region1 = 24
delta_z_region1 =  0.015625

# Parameters for region 2
fuel_id2 = 301
clad_id2 = 302
mod_id2  = 303
cr_id2   = 304
axial_cells_region2 = 48
delta_z_region2 = 0.8671875

# Parameters for region 2
fuel_id3 = 401
clad_id3 = 402
```

```
mod_id3  = 403
cr_id3   = 404
axial_cells_region3 = 48
delta_z_region3 =  0.8671875

Nz = axial_cells_region0 + axial_cells_region1 + axial_cells_region2 +
axial_cells_region3

##----------------------------------------------------------------------------##
## BUILD MESH
##----------------------------------------------------------------------------##
print "Defining build_mesh"

def build_mesh(N):
    # uniform layout for reflector/plate regions
    uniform_layout = [1, 1, 1,
                      1, 1, 1,
                      1, 1, 1]

    # fuel lattice arrangement
    fuel_layout = [1, 1, 1,
                   1, 2, 1,
                   1, 1, 1]

    # Bottom reflector
    lattice = []
    for k in xrange(num_mat_levels):
        lattice[k:] = [Array_0(lat_size)]

    # Number of clean materials per level
    num_mat = [0] * num_mat_levels

    # Number of clean materials total
    num_clean_mat = 0

    # Bottom region
    print "Building level  0"
    pin = Pincell()
    pin.set_pitch( pitch )
    r = [clad_in, clad_out]
    ids = [ fuel_id0, clad_id0 ]
    pin.set_shells(ids, r, mod_id0)

    gt = Pincell()
    gt.set_pitch( pitch )
    r = [gt_in, gt_out]
    ids = [mod_id0, clad_id0]
    gt.set_shells(ids, r, mod_id0)

    # Assign reflector pin to bottom level of lattice
    lattice[0].set_objects(fuel_layout)
    lattice[0].assign_object(pin,1)
    lattice[0].assign_object(gt,2)
    lattice[0].build_array(N,0)

    # Set global mixing table
    num_mat[0] = lattice[0].num_mat()
    mix_table = lattice[0].mixing_vector()
    num_clean_mat = num_mat[0]
```

```
# Create all other levels
for k in xrange(1,num_mat_levels):
    print "Building level ",k

    # Fuel regions
    if k==1:
        pin = Pincell()
        pin.set_pitch( pitch )
        r = [clad_in, clad_out]
        ids = [fuel_id1, clad_id1]
        pin.set_shells(ids, r, mod_id1)

        gt = Pincell()
        gt.set_pitch( pitch )
        r = [gt_in, gt_out]
        ids = [cr_id1, clad_id1]
        gt.set_shells(ids, r, mod_id1)

        # Build lattice for this level
        lattice[k].set_objects(fuel_layout)
        lattice[k].assign_object(pin,1)
        lattice[k].assign_object(gt,2)

    # Fuel regions
    if k==2:
        pin = Pincell()
        pin.set_pitch( pitch )
        r = [clad_in, clad_out]
        ids = [fuel_id2, clad_id2]
        #ids = [cr_id1, clad_id1]
        pin.set_shells(ids, r, mod_id2)

        gt = Pincell()
        gt.set_pitch( pitch )
        r = [gt_in, gt_out]
        #ids = [fuel_id1, clad_id1]
        ids = [cr_id2, clad_id2]
        gt.set_shells(ids, r, mod_id2)

        # Build lattice for this level
        lattice[k].set_objects(fuel_layout)
        lattice[k].assign_object(pin,1)
        lattice[k].assign_object(gt,2)

    # Fuel regions
    if k==3:
        pin = Pincell()
        pin.set_pitch( pitch )
        r = [clad_in, clad_out]
        ids = [fuel_id3, clad_id3]
        #ids = [cr_id1, clad_id1]
        pin.set_shells(ids, r, mod_id3)

        gt = Pincell()
        gt.set_pitch( pitch )
        r = [gt_in, gt_out]
        #ids = [fuel_id1, clad_id1]
        ids = [cr_id3, clad_id3]
        gt.set_shells(ids, r, mod_id3)
        # Build lattice for this level
        lattice[k].set_objects(fuel_layout)
        lattice[k].assign_object(pin,1)
        lattice[k].assign_object(gt,2)

    # Finish lattice construction for this level
    lattice[k].build_array(N,N,0,mix_table,num_clean_mat)

    # Update global mixing table
    num_mat[k] = lattice[k].num_mat()
    mix_table  = lattice[k].mixing_vector()

    if num_mat[k]>num_clean_mat:
        num_clean_mat = num_mat[k]

    # Update lower level matids
    for j in xrange(k):
        lattice[j].set_mixids( lattice[k].update_old_matids( \
            lattice[j].num_mat(), lattice[j].mixids() ) )
        lattice[j].set_num_mat( num_clean_mat )

# Build axial mesh
for k in xrange(Nz):
    if k < axial_cells_region1:
        z[k+1] = z[k] + delta_z_region0
    if (k >=axial_cells_region1 and k < axial_cells_region2):
        z[k+1] = z[k] + delta_z_region1
    if (k >=axial_cells_region2 and k < axial_cells_region3):
        z[k+1] = z[k] + delta_z_region2
    if (k >axial_cells_region3):
        z[k+1] = z[k] + delta_z_region3
# mesh planes in x,y
xy  = lattice[0].xy_planes()

# Set up mixture ids for all cells
mixids = Vec_Int(Nz*Ny*Nx, 0)

# Lower material region
offset = 0
for local_k in xrange(axial_cells_region0):
    k = local_k + offset
    xyids = lattice[0].mixids()
    for j in xrange(Ny):
        for i in xrange(Nx):
            cell = i + Nx * (j + k*Ny)
            plane_cell = i + Nx*j
            mixids[cell] = xyids[plane_cell]

# Upper material region
offset = axial_cells_region0
for local_k in xrange(axial_cells_region1):
    k = local_k + offset
    xyids = lattice[1].mixids()
    for j in xrange(Ny):
        for i in xrange(Nx):
            cell = i + Nx * (j + k*Ny)
            plane_cell = i + Nx*j
            mixids[cell] = xyids[plane_cell]

# Upper material region - Control rod In
```

```
    offset = axial_cells_region0 + axial_cells_region1
    for local_k in xrange(axial_cells_region2):
        k = local_k + offset
        xyids = lattice[2].mixids()
        for j in xrange(Ny):
            for i in xrange(Nx):
                cell = i + Nx * (j + k*Ny)
                plane_cell = i + Nx*j
                mixids[cell] = xyids[plane_cell]

    # Upper material region - Control rod In
    offset = axial_cells_region0 + axial_cells_region1 + axial_cells_region2
    for local_k in xrange(axial_cells_region3):
        k = local_k + offset
        xyids = lattice[3].mixids()
        for j in xrange(Ny):
            for i in xrange(Nx):
                cell = i + Nx * (j + k*Ny)
                plane_cell = i + Nx*j
                mixids[cell] = xyids[plane_cell]

# list of ids
    ids    = Vec_Int(num_clean_mat)
    for m in xrange(len(ids)):
        ids[m] = m

    return (xy, z, mixids, ids, mix_table, lattice)

##---------------------------------------------------------------------------##
## MAIN
##---------------------------------------------------------------------------##

initialize(sys.argv)

if node() == 0:
    print "Denovo - pykba Python Front-End"
    print "-------------------------------"
    print "Release      : %16s" % (release())
    print "Release Date : %16s" % (release_date())
    print "Build Date   : %16s" % (build_date())
    print

timer = Timer()
timer.start()

##---------------------------------------------------------------------------##
## DB
##---------------------------------------------------------------------------##

db = DB("pykba")

# problem type
db.insert("problem_type", "FIXED_SOURCE")

db.insert("num_z_blocks", 1)
# decomposition
if nodes() == 1:

    db.insert("num_blocks_i", 1)
    db.insert("num_blocks_j", 1)
    iblocks = 1
    jblocks = 1

elif nodes() == 2:

    db.insert("num_blocks_i", 2)
    db.insert("num_blocks_j", 1)
    iblocks = 2
    jblocks = 1

elif nodes() == 4:

    db.insert("num_blocks_i", 2)
    db.insert("num_blocks_j", 2)
    iblocks = 2
    jblocks = 2

elif nodes() == 8:

    db.insert("num_blocks_i", 4)
    db.insert("num_blocks_j", 2)
    iblocks = 4
    jblocks = 2

elif nodes() == 16:

    db.insert("num_blocks_i", 4)
    db.insert("num_blocks_j", 4)

elif nodes() == 32:

    db.insert("num_blocks_i", 8)
    db.insert("num_blocks_j", 4)

elif nodes() == 64:

    db.insert("num_blocks_i", 8)
    db.insert("num_blocks_j", 8)

#db.insert("num_blocks_i", 1)
#db.insert("num_blocks_j", 1)
db.insert("num_sets",     1)

# energy partitioning
db.insert("partition_upscatter", 1, 1)

# data settings
db.insert("num_groups", Ng)
db.insert("downscatter", 0, 1)
db.insert("Pn_order", pn_order)

# solver setup
#db.insert("eigen_solver", "arnoldi")
db.insert("mg_solver", "krylov")
#db.insert("within_group_solver","GMRES_R")
db.insert("tolerance", 1.0e-5)
db.insert("aztec_kspace", 50)
db.insert("max_itr", 50)

db.insert("iterate_downscatter", store_bound, 1)
db.insert("use_init_guess", store_bound, 1)

# eigenvalue information
db.add_db("eigenvalue_db", "eigenvalue")
db.insert("eigenvalue_db", "L2_tolerance", 1e-5)
#db.insert("eigenvalue_db", "k_tolerance", 1e-5)
db.insert("eigenvalue_db", "diagnostic_level", 2)
db.insert("eigenvalue_db", "keff", 1.0)
db.insert("eigenvalue_db", "inner_tol_relaxer", "CONSTANT")
db.insert("eigenvalue_db", "inner_tol_relax_factor",1.5)
db.insert("eigenvalue_db", "arnoldi_restarts", 10)
db.insert("eigenvalue_db", "arnoldi_kspace", 15)
db.insert("eigenvalue_db", "energy_dep_ev", 1, 1)
db.insert("eigenvalue_db", "calculate_moments", 1, 1)

# upscatter database
db.add_db("upscatter_db", "upscatter")
db.insert("upscatter_db", "tolerance", 1.0e-6)
db.insert("upscatter_db", "aztec_kspace", 250)
db.insert("upscatter_db", "aztec_diag", 1)
db.insert("upscatter_db", "aztec_output", 1)

# Mesh
(x, z, matids, cleanids, table, lattice) = build_mesh(Npin)
print "Done with build_mesh()"
db.insert("x_edges", x)
db.insert("y_edges", x)
db.insert("z_edges", z)

## Boundary conditions
bounds = [1, 1, 1, 1, 0, 0]
#bounds = [0, 0, 0, 0, 0, 0]
db.insert("boundary", "reflect")
db.add_db("boundary_db", "bnd_conditions")
db.insert("boundary_db", "reflect", bounds, 1)
db.insert("boundary_db", "store_bnd_state", store_bound, 1)

# Angular options
db.add_db("quadrature_db", "quad_options")
#db.insert("quadrature_db", "quad_type", "qr")
#db.insert("quadrature_db", "quad_type", "ldfe")
#db.insert("quadrature_db", "order", 3)
db.insert("quadrature_db", "Sn_order", 6)
#db.insert("quadrature_db", "quad_type", "glproduct")
#db.insert("quadrature_db", "polars_octant", 2)
#db.insert("quadrature_db", "azimuthals_octant",2)

##---------------------------------------------------------------------------##
## MANAGER
##---------------------------------------------------------------------------##

# make manager, material, and angles
manager = Manager()
mat     = Mat()
angles  = Angles()

# partition the problem
print "Manager partitioning"
manager.partition(db, mat, angles)
print "Manager done partitioning"
```

```
# get mapping and mesh objects
mapp    = manager.get_map()
indexer = manager.get_indexer()
mesh    = manager.get_mesh()

# global and local cell numbers
Gx = indexer.num_global(X)
Gy = indexer.num_global(Y)
Gz = mesh.num_cells_dim(Z)
Nx = mesh.num_cells_dim(X)
Ny = mesh.num_cells_dim(Y)
Nz = mesh.num_cells_dim(Z)

if node() == 0:
    print ">>> Partitioned global mesh with %i x %i x %i cells" \
          % (Gx, Gy, Gz)

##-----------------------------------------------------------------------------##
## MATERIAL SETUP
##-----------------------------------------------------------------------------##

# AMPX library
ampx = AMPX()
ampx.read_AMPX("Halden-CSAS.lib")

xsdb = XS_DB(db)
xsdb.set_num(405)
xsdb.assign_ampx(fuel_id0, 1, ampx)
xsdb.assign_ampx(fuel_id1, 1, ampx)
xsdb.assign_ampx(fuel_id2, 1, ampx)
xsdb.assign_ampx(fuel_id3, 1, ampx)
xsdb.assign_ampx(clad_id0, 2, ampx)
xsdb.assign_ampx(clad_id1, 2, ampx)
xsdb.assign_ampx(clad_id2, 2, ampx)
xsdb.assign_ampx(clad_id3, 2, ampx)
xsdb.assign_ampx(mod_id0,  4, ampx)
xsdb.assign_ampx(mod_id1,  4, ampx)
xsdb.assign_ampx(mod_id2,  4, ampx)
xsdb.assign_ampx(mod_id3,  4, ampx)
xsdb.assign_ampx(cr_id1,   4, ampx)
xsdb.assign_ampx(cr_id2,   5, ampx)
xsdb.assign_ampx(cr_id3,   5, ampx)

beta_t = .007249
#normal_keff =1.07039043
normal_keff = 1.044

print 'Creating New XSDB on Node ',node()
xsdb_mod = XS_DB(db)
xsdb_mod.set_num(xsdb.num_mat())
for imat in xrange(xsdb.num_mat()):
    if( xsdb.assigned(imat) ):
        xs_chi    = [0.0]*Ng
        xs_nusigf = [0.0]*Ng
        for g in xrange(Ng):
            if( xsdb.assigned_fission(imat) ):
                xs_chi[g] = xsdb.fission_data(imat,g,CHI)
                xs_nusigf[g] = xsdb.fission_data(imat,g,NU_SIGMA_F)/normal_keff
            total_g = xsdb.total(imat,g)
            scatter_g = []
            cols = [0]*(Ng-g-1)
            for gp in xrange(Ng):
                scatter_g += [[0.0]]
                if( gp>g ):
                    cols[gp-g-1] = gp
                if( gp<=g or xsdb.has_upscatter(imat,g,gp) ):
                    scatter_g[gp] = [xsdb.scatter(imat,g,gp,0)]
                    for ipn in xrange(1,pn_order+1):
                        scatter_g[gp] += [xsdb.scatter(imat,g,gp,ipn)]
                if xsdb.assigned_fission(imat):
                    scatter_g[gp][0] += ((1.0-
beta_t)/normal_keff)*xsdb.fission_data(imat,g,CHI)*xsdb.fission_data(imat,gp,NU_SIGMA_
F)
            xsdb_mod.assign_upscatter(imat,g,total_g,cols,scatter_g)
        if( xsdb.assigned_fission(imat) ):
            xsdb_mod.assign_fission(imat,xs_nusigf,xs_chi)

##---------------------------------------------------------------------------##
## DELAYED GROUPS
##---------------------------------------------------------------------------##

beta=Vec_Dbl(6,0.0)

beta[0]=2.43e-04
beta[1]=1.45E-03
beta[2]=1.34e-03
beta[3]=2.92e-03
beta[4]=1.04e-03
beta[5]=2.56e-04

lamb=Vec_Dbl(6,0.0)

lamb[0]= 1.27e-02
lamb[1]= 3.17e-02
lamb[2]= 1.17e-01
lamb[3]= 3.14e-01
lamb[4]= 1.38
lamb[5]= 3.83

# make macro mixer
mixer = Macro_Mixer(xsdb_mod)
mixer.set(cleanids, table)

# make the material database
mixer.mix_with_global_ids(matids, mat)

##---------------------------------------------------------------------------##
## ENERGY PARTITIONING
##---------------------------------------------------------------------------##

print "Partitioning energy"
manager.partition_energy(mat, angles)
erg_set = manager.get_erg_set_comm()

##---------------------------------------------------------------------------##
## SOURCE SETUP
##---------------------------------------------------------------------------##

# allocate problem state (using Isotropic source)
```

```
print "Allocating state"
source = Isotropic_Source()

# Read source term from file and assign to source object
num_files = 64
ids  = Vec_Int(mapp.num_global(), 0)
qext = Vec_Dbl(mapp.num_global())
for ifile in xrange(num_files):
    thisfile = open('precursors'+str(ifile)+'.txt','r')
    firstline = thisfile.readline()
    numcells_file = int(firstline.strip())
    for isrc in xrange(numcells_file):
        thisline = thisfile.readline()
        thisline = thisline.strip()
        thisline = thisline.split()
        global_cell = int(thisline[0])
        src_cell    = float(thisline[1])
        qext[global_cell] = src_cell / normal_keff
    thisfile.close()

print 'Done reading source on ',node()

spectrum = Vec_Dbl(Ng)
for g in xrange(Ng):
    spectrum[g]=xsdb.fission_data(fuel_id0,g,CHI)

# Set up manager
print "Setting up manager"
manager.setup(source)

# Set the source
source.set(1,spectrum,ids,qext)
##---------------------------------------------------------------------------##
## SOLVE
##---------------------------------------------------------------------------##

if node() == 0:
    print ">>> Setup complete"
    print ">>> Solving with %s differencing and %s quadrature (%i angles)" \
          % (manager.spatial_descriptor(), angles.quad_label(), angles.num_angles())

if run_problem==1:

    print "Running problem"

    # solve the problem
    manager.solve(angles)

    steady_precursors = Vec_Dbl(mesh.num_cells(),0.0)

    power = Vec_Dbl(mesh.num_cells(),0.0)
    for cell in xrange(mesh.num_cells()):
        matid = mat.matid(cell)
        if mat.assigned_fission(matid):
            for g in xrange(Ng):
                phi = Moments(g)
                power[cell] +=  phi.scalar_flux(cell) \
                        * mat.fission_data(matid, g, NU_SIGMA_F)

    for cell in xrange(mesh.num_cells()):
```

```
    for i in xrange(6):
        steady_precursors[cell] = steady_precursors[cell] + power[i]*beta[i]

filename= open('precursors'+str(node())+'_step2.txt','w')
for cell in xrange(mesh.num_cells()):
    filename.write(str(steady_precursors[cell])+'\n')
filename.close()

flux0 = Vec_Dbl(mesh.num_cells(), 0.0)
flux5 = Vec_Dbl(mesh.num_cells(), 0.0)
flux10 = Vec_Dbl(mesh.num_cells(), 0.0)
flux15 = Vec_Dbl(mesh.num_cells(), 0.0)
flux20 = Vec_Dbl(mesh.num_cells(), 0.0)
flux25 = Vec_Dbl(mesh.num_cells(), 0.0)
flux30 = Vec_Dbl(mesh.num_cells(), 0.0)
flux35 = Vec_Dbl(mesh.num_cells(), 0.0)
flux40 = Vec_Dbl(mesh.num_cells(), 0.0)

phi = Moments(0)
for cell in xrange(mesh.num_cells()):
    flux0[cell]  = phi.scalar_flux(cell)

phi = Moments(5)
for cell in xrange(mesh.num_cells()):
    flux5[cell]  = phi.scalar_flux(cell)

phi = Moments(10)
for cell in xrange(mesh.num_cells()):
    flux10[cell] = phi.scalar_flux(cell)

phi = Moments(15)
for cell in xrange(mesh.num_cells()):
    flux15[cell] = phi.scalar_flux(cell)

phi = Moments(20)
for cell in xrange(mesh.num_cells()):
    flux20[cell] = phi.scalar_flux(cell)

phi = Moments(25)
for cell in xrange(mesh.num_cells()):
    flux25[cell] = phi.scalar_flux(cell)

phi = Moments(30)
for cell in xrange(mesh.num_cells()):
    flux30[cell] = phi.scalar_flux(cell)

phi = Moments(35)
for cell in xrange(mesh.num_cells()):
    flux35[cell] = phi.scalar_flux(cell)

phi = Moments(40)
for cell in xrange(mesh.num_cells()):
    flux40[cell] = phi.scalar_flux(cell)

silo = SILO()
silo.add_mixer(mixer)

silo.open(str(lat_size)+"x"+str(lat_size)+"_r"+str(Npin)+"a"+str(Nz)+'fixed')

silo.add("flux0",  flux0)
```

```
    silo.add("flux5",  flux5)
    silo.add("flux10", flux10)
    silo.add("flux15", flux15)
    silo.add("flux20", flux20)
    silo.add("flux25", flux25)
    silo.add("flux30", flux30)
    silo.add("flux35", flux35)
    silo.add("flux40", flux40)
    silo.add("power",  power)

    silo.close()

# Write HPCKBA input
if generate_hpckba==1:

    print "Generating HPCKBA input"

    out = HPC_Problem_Output(1, Nx, Ny, Nz)

    print out.chunk(), out.num_chunks_per_file(), out.num_files()

    out.open(str(lat_size)+"x"+str(lat_size)+"_r"+str(Npin)+"a"+str(Nz))

    shapes = Vec_Dbl()

    out.write_db(db)
    out.write_mixed_xs(mixer)
    out.write_src_info(ZERO_SOURCE, ZERO_SOURCE, shapes)

    # write matids
    out.start_field_loop()
    ids = Vec_Int(out.chunk(), 0)
    while not out.finished_field_loop():
        k   = out.current_chunk()
        if k < Nz:
           print "Writing k-plane %d/%d" % (k, Nz)
           for j in xrange(Gy):
               for i in xrange(Gx):
                   cell  = indexer.g2g(i,j,k)
                   index = indexer.g2g(i,j,0)
                   ids[index] = matids[cell]
        out.write_matids(ids)
        out.advance_loop()

    out.close()
##---------------------------------------------------------------------------##
## TIMING
##---------------------------------------------------------------------------##

# output final database (has class-dependent defaults)
db.output()

timer.stop()
time = timer.wall_clock()

keys = timer_keys()
if len(keys) > 0 and node() == 0:
    print "\n"
    print "TIMING : Problem ran in %16.6e seconds." % (time)
    print "------------------------------------------------"
    for key in keys:
        print "%30s : %16.6e" % (key, timer_value(key) / time)
    print "---------------------------------------------------"

##---------------------------------------------------------------------------##

manager.close()
finalize()

###############################################################################
## end
###############################################################################
```

DENOVO Fixed Source Step 2 – Halden Case

```
###############################################################################
##
## HALDEN 3x3 - CRout Step 2
##
###############################################################################
## Copyright (C) 2008 Oak Ridge National Laboratory, UT-Battelle, LLC.
##---------------------------------------------------------------------------##
## generated by /data/denovo/build/debug/bin/pygen built on 20101129
###############################################################################

import os, sys, math, string

# pykba equation type
from sc import *
unks_cell = 1

Npin = 8 # number of cells per pin in coarse mesh
Ng = 44   #number of energy groups on the AMPX library
Nz = 40
Zmax = 400.
pn_order = 0

store_bound = 1

run_problem     = 1
generate_hpckba = 0

# Starting z value (this places bottom of fuel at z=0)
Zmin        = 0.0

# Number of different axial material levels
num_mat_levels = 3

# Pin spacing
pitch = 1.4224

# Clad inner/outer radius
clad_in  = 0.545465
clad_out = 0.639445

# Guid tube inner/outer radius
```

```
gt_in  = 0.569
gt_out = 0.6448

lat_size = 3
Nx = lat_size*Npin
Ny = lat_size*Npin
plane_cells = Nx*Ny

store_bound = 0

run_problem     = 1
generate_hpckba = 0


# Parameters for region 0
fuel_id0 = 101
clad_id0 = 102
mod_id0  = 103
axial_cells_region0 = 64
delta_z_region0 = 5.541015625

# Parameters for region 1
fuel_id1 = 201
clad_id1 = 202
mod_id1  = 203
cr_id1   = 204
axial_cells_region1 = 24
delta_z_region1 =  0.15625

# Parameters for region 2
fuel_id2 = 301
clad_id2 = 302
mod_id2  = 303
cr_id2   = 304
axial_cells_region2 = 48
delta_z_region2 =  0.8671875

Nz = axial_cells_region0 + axial_cells_region1 + axial_cells_region2

##----------------------------------------------------------------------------##
## BUILD MESH
##----------------------------------------------------------------------------##
print "Defining build_mesh"

def build_mesh(N):
    # uniform layout for reflector/plate regions
    uniform_layout = [1, 1, 1,
                      1, 1, 1,
                      1, 1, 1]

    # fuel lattice arrangement
    fuel_layout = [1, 1, 1,
                   1, 2, 1,
                   1, 1, 1]

    # Bottom reflector
    lattice = []
    for k in xrange(num_mat_levels):
        lattice[k:] = [Array_0(lat_size)]
```

```
# Number of clean materials per level
num_mat = [0] * num_mat_levels

# Number of clean materials total
num_clean_mat = 0

# Bottom region
print "Building level  0"
pin = Pincell()
pin.set_pitch( pitch )
r = [clad_in, clad_out]
ids = [ fuel_id0, clad_id0 ]
pin.set_shells(ids, r, mod_id0)

gt = Pincell()
gt.set_pitch( pitch )
r = [gt_in, gt_out]
ids = [mod_id0, clad_id0]
gt.set_shells(ids, r, mod_id0)

# Assign reflector pin to bottom level of lattice
lattice[0].set_objects(fuel_layout)
lattice[0].assign_object(pin,1)
lattice[0].assign_object(gt,2)
lattice[0].build_array(N,0)

# Set global mixing table
num_mat[0] = lattice[0].num_mat()
mix_table = lattice[0].mixing_vector()
num_clean_mat = num_mat[0]

# Create all other levels
for k in xrange(1,num_mat_levels):
    print "Building level ",k

    # Fuel regions
    if k==1:
        pin = Pincell()
        pin.set_pitch( pitch )
        r = [clad_in, clad_out]
        ids = [fuel_id1, clad_id1]
        pin.set_shells(ids, r, mod_id1)

        gt = Pincell()
        gt.set_pitch( pitch )
        r = [gt_in, gt_out]
        ids = [cr_id1, clad_id1]
        gt.set_shells(ids, r, mod_id1)

        # Build lattice for this level
        lattice[k].set_objects(fuel_layout)
        lattice[k].assign_object(pin,1)
        lattice[k].assign_object(gt,2)

    # Fuel regions
    if k==2:
        pin = Pincell()
        pin.set_pitch( pitch )
        r = [clad_in, clad_out]
        ids = [fuel_id2, clad_id2]
```

```
            #ids = [cr_id1, clad_id1]
            pin.set_shells(ids, r, mod_id2)

            gt = Pincell()
            gt.set_pitch( pitch )
            r = [gt_in, gt_out]
            #ids = [fuel_id1, clad_id1]
            ids = [cr_id2, clad_id2]
            gt.set_shells(ids, r, mod_id2)

            # Build lattice for this level
            lattice[k].set_objects(fuel_layout)
            lattice[k].assign_object(pin,1)
            lattice[k].assign_object(gt,2)

        # Finish lattice construction for this level
        lattice[k].build_array(N,N,0,mix_table,num_clean_mat)

        # Update global mixing table
        num_mat[k] = lattice[k].num_mat()
        mix_table  = lattice[k].mixing_vector()

        if num_mat[k]>num_clean_mat:
            num_clean_mat = num_mat[k]

        # Update lower level matids
        for j in xrange(k):
            lattice[j].set_mixids( lattice[k].update_old_matids( \
                lattice[j].num_mat(), lattice[j].mixids() ) )
            lattice[j].set_num_mat( num_clean_mat )

    # Build axial mesh
    z = [0.0] * (Nz + 1)
    z[0] = Zmin
    for k in xrange(Nz):
        if k < axial_cells_region0:
            z[k+1] = z[k] + delta_z_region0
        elif (k >=axial_cells_region0 and k <
(axial_cells_region0+axial_cells_region1)):
            z[k+1] = z[k] + delta_z_region1
        else:
            z[k+1] = z[k] + delta_z_region2

    # mesh planes in x,y
    xy  = lattice[0].xy_planes()

    # Set up mixture ids for all cells
    mixids = Vec_Int(Nz*Ny*Nx, 0)

    # Lower material region
    offset = 0
    for local_k in xrange(axial_cells_region0):
        k = local_k + offset
        xyids = lattice[0].mixids()
        for j in xrange(Ny):
            for i in xrange(Nx):
                cell = i + Nx * (j + k*Ny)
                plane_cell = i + Nx*j
                mixids[cell] = xyids[plane_cell]
    # Upper material region
    offset = axial_cells_region0
    for local_k in xrange(axial_cells_region1):
        k = local_k + offset
        xyids = lattice[1].mixids()
        for j in xrange(Ny):
            for i in xrange(Nx):
                cell = i + Nx * (j + k*Ny)
                plane_cell = i + Nx*j
                mixids[cell] = xyids[plane_cell]

    # Upper material region - Control rod In
    offset = axial_cells_region0 + axial_cells_region1
    for local_k in xrange(axial_cells_region2):
        k = local_k + offset
        xyids = lattice[2].mixids()
        for j in xrange(Ny):
            for i in xrange(Nx):
                cell = i + Nx * (j + k*Ny)
                plane_cell = i + Nx*j
                mixids[cell] = xyids[plane_cell]

# list of ids
    ids    = Vec_Int(num_clean_mat)
    for m in xrange(len(ids)):
        ids[m] = m

    return (xy, z, mixids, ids, mix_table, lattice)

##---------------------------------------------------------------------------##
## MAIN
##---------------------------------------------------------------------------##

initialize(sys.argv)

if node() == 0:
    print "Denovo - pykba Python Front-End"
    print "-------------------------------"
    print "Release      : %16s" % (release())
    print "Release Date : %16s" % (release_date())
    print "Build Date   : %16s" % (build_date())
    print

timer = Timer()
timer.start()

##---------------------------------------------------------------------------##
## DB
##---------------------------------------------------------------------------##

db = DB("pykba")

# problem type
db.insert("problem_type", "FIXED_SOURCE")

db.insert("num_z_blocks", 1)
# decomposition
if nodes() == 1:

    db.insert("num_blocks_i", 1)
```

```
        db.insert("num_blocks_j", 1)
        iblocks = 1
        jblocks = 1

    elif nodes() == 2:

        db.insert("num_blocks_i", 2)
        db.insert("num_blocks_j", 1)
        iblocks = 2
        jblocks = 1

    elif nodes() == 4:

        db.insert("num_blocks_i", 2)
        db.insert("num_blocks_j", 2)
        iblocks = 2
        jblocks = 2

    elif nodes() == 8:

        db.insert("num_blocks_i", 4)
        db.insert("num_blocks_j", 2)
        iblocks = 4
        jblocks = 2

    elif nodes() == 16:

        db.insert("num_blocks_i", 4)
        db.insert("num_blocks_j", 4)

    elif nodes() == 32:

        db.insert("num_blocks_i", 8)
        db.insert("num_blocks_j", 4)

    elif nodes() == 64:

        db.insert("num_blocks_i", 8)
        db.insert("num_blocks_j", 8)

    #db.insert("num_blocks_i", 1)
    #db.insert("num_blocks_j", 1)
    db.insert("num_sets",     1)

    # energy partitioning
    db.insert("partition_upscatter", 1, 1)

    # data settings
    db.insert("num_groups", Ng)
    db.insert("downscatter", 0, 1)
    db.insert("Pn_order", pn_order)

    # solver setup
    #db.insert("eigen_solver", "arnoldi")
    db.insert("mg_solver", "krylov")
    #db.insert("within_group_solver","GMRES_R")
    db.insert("tolerance", 1.0e-5)
    db.insert("aztec_kspace", 250)
    db.insert("max_itr", 250)
```

```
db.insert("iterate_downscatter", store_bound, 1)
db.insert("use_init_guess", store_bound, 1)

# eigenvalue information
db.add_db("eigenvalue_db", "eigenvalue")
db.insert("eigenvalue_db", "L2_tolerance", 1e-5)
#db.insert("eigenvalue_db", "k_tolerance", 1e-5)
db.insert("eigenvalue_db", "diagnostic_level", 2)
db.insert("eigenvalue_db", "keff", 1.0)
db.insert("eigenvalue_db", "inner_tol_relaxer", "CONSTANT")
db.insert("eigenvalue_db", "inner_tol_relax_factor",1.5)
db.insert("eigenvalue_db", "arnoldi_restarts", 10)
db.insert("eigenvalue_db", "arnoldi_kspace", 15)
db.insert("eigenvalue_db", "energy_dep_ev", 1, 1)
db.insert("eigenvalue_db", "calculate_moments", 1, 1)

# upscatter database
db.add_db("upscatter_db", "upscatter")
db.insert("upscatter_db", "tolerance", 1.0e-6)
db.insert("upscatter_db", "aztec_kspace", 250)
db.insert("upscatter_db", "aztec_diag", 1)
db.insert("upscatter_db", "aztec_output", 1)

# Mesh
(x, z, matids, cleanids, table, lattice) = build_mesh(Npin)
print "Done with build_mesh()"
db.insert("x_edges", x)
db.insert("y_edges", x)
db.insert("z_edges", z)

## Boundary conditions
bounds = [1, 1, 1, 1, 0, 0]
#bounds = [0, 0, 0, 0, 0, 0]
db.insert("boundary", "reflect")
db.add_db("boundary_db", "bnd_conditions")
db.insert("boundary_db", "reflect", bounds, 1)
db.insert("boundary_db", "store_bnd_state", store_bound, 1)

# Angular options
db.add_db("quadrature_db", "quad_options")
#db.insert("quadrature_db", "quad_type", "qr")
#db.insert("quadrature_db", "quad_type", "ldfe")
#db.insert("quadrature_db", "order", 3)
db.insert("quadrature_db", "Sn_order", 6)
#db.insert("quadrature_db", "quad_type", "glproduct")
#db.insert("quadrature_db", "polars_octant", 2)
#db.insert("quadrature_db", "azimuthals_octant",2)

##---------------------------------------------------------------------------##
## MANAGER
##---------------------------------------------------------------------------##

# make manager, material, and angles
manager = Manager()
mat     = Mat()
angles  = Angles()

# partition the problem
print "Manager partitioning"
manager.partition(db, mat, angles)
```

```
print "Manager done partitioning"

# get mapping and mesh objects
mapp    = manager.get_map()
indexer = manager.get_indexer()
mesh    = manager.get_mesh()

# global and local cell numbers
Gx = indexer.num_global(X)
Gy = indexer.num_global(Y)
Gz = mesh.num_cells_dim(Z)
Nx = mesh.num_cells_dim(X)
Ny = mesh.num_cells_dim(Y)
Nz = mesh.num_cells_dim(Z)

if node() == 0:
    print ">>> Partitioned global mesh with %i x %i x %i cells" \
          % (Gx, Gy, Gz)

##---------------------------------------------------------------------------##
## MATERIAL SETUP
##---------------------------------------------------------------------------##

# AMPX library
ampx = AMPX()
ampx.read_AMPX("Halden-CSAS.lib")

xsdb = XS_DB(db)
xsdb.set_num(405)
xsdb.assign_ampx(fuel_id0, 1, ampx)
xsdb.assign_ampx(fuel_id1, 1, ampx)
xsdb.assign_ampx(fuel_id2, 1, ampx)
xsdb.assign_ampx(clad_id0, 2, ampx)
xsdb.assign_ampx(clad_id1, 2, ampx)
xsdb.assign_ampx(clad_id2, 2, ampx)
xsdb.assign_ampx(mod_id0,  4, ampx)
xsdb.assign_ampx(mod_id1,  4, ampx)
xsdb.assign_ampx(mod_id2,  4, ampx)
xsdb.assign_ampx(cr_id1,   4, ampx)
xsdb.assign_ampx(cr_id2,   5, ampx)

beta_t = .007249
normal_keff = 1.0839975787

print 'Creating New XSDB on Node ',node()
xsdb_mod = XS_DB(db)
xsdb_mod.set_num(xsdb.num_mat())
for imat in xrange(xsdb.num_mat()):
    if( xsdb.assigned(imat) ):
        xs_chi    = [0.0]*Ng
        xs_nusigf = [0.0]*Ng
        for g in xrange(Ng):
            if( xsdb.assigned_fission(imat) ):
                xs_chi[g] = xsdb.fission_data(imat,g,CHI)
                xs_nusigf[g] = xsdb.fission_data(imat,g,NU_SIGMA_F)
                xs_nusigf[g] = xs_nusigf[g]/normal_keff
            total_g = xsdb.total(imat,g)
            scatter_g = []
            cols = [0]*(Ng-g-1)
            for gp in xrange(Ng):
                scatter_g += [[0.0]]
                if( gp>g ):
                    cols[gp-g-1] = gp
                if( gp<=g or xsdb.has_upscatter(imat,g,gp) ):
                    scatter_g[gp] = [xsdb.scatter(imat,g,gp,0)]
                    for ipn in xrange(1,pn_order+1):
                        scatter_g[gp] += [xsdb.scatter(imat,g,gp,ipn)]
                if xsdb.assigned_fission(imat):
                    scatter_g[gp][0] += ((1.0-
beta_t)/normal_keff)*xsdb.fission_data(imat,g,CHI)*xsdb.fission_data(imat,gp,NU_SIGMA_
F)
            xsdb_mod.assign_upscatter(imat,g,total_g,cols,scatter_g)
        if( xsdb.assigned_fission(imat) ):
            xsdb_mod.assign_fission(imat,xs_nusigf,xs_chi)

##---------------------------------------------------------------------------##
## DELAYED GROUPS
##---------------------------------------------------------------------------##

beta=Vec_Dbl(6,0.0)

beta[0]=2.43e-04
beta[1]=1.45E-03
beta[2]=1.34e-03
beta[3]=2.92e-03
beta[4]=1.04e-03
beta[5]=2.56e-04

lamb=Vec_Dbl(6,0.0)

lamb[0]= 1.27e-02
lamb[1]= 3.17e-02
lamb[2]= 1.17e-01
lamb[3]= 3.14e-01
lamb[4]= 1.38
lamb[5]= 3.83

# make macro mixer
mixer = Macro_Mixer(xsdb_mod)
mixer.set(cleanids, table)

# make the material database
mixer.mix_with_global_ids(matids, mat)

##---------------------------------------------------------------------------##
## ENERGY PARTITIONING
##---------------------------------------------------------------------------##

print "Partitioning energy"
manager.partition_energy(mat, angles)
erg_set = manager.get_erg_set_comm()

##---------------------------------------------------------------------------##
## SOURCE SETUP
##---------------------------------------------------------------------------##

# allocate problem state (using Isotropic source)
print "Allocating state"
source = Isotropic_Source()
```

```
# Read source term from file and assign to source object
num_files = 64
ids  = Vec_Int(mapp.num_global(), 0)
qext = Vec_Dbl(mapp.num_global())
for ifile in xrange(num_files):
    thisfile = open('precursors'+str(ifile)+'_step2.txt','r')
    firstline = thisfile.readline()
    numcells_file = int(firstline.strip())
    for isrc in xrange(numcells_file):
        thisline = thisfile.readline()
        thisline = thisline.strip()
        thisline = thisline.split()
        global_cell = int(thisline[0])
        src_cell    = float(thisline[1])
        qext[global_cell] = src_cell
    thisfile.close()

print 'Done reading source on ',node()

spectrum = Vec_Dbl(Ng)
for g in xrange(Ng):
    spectrum[g]=xsdb.fission_data(fuel_id0,g,CHI)

# Set up manager
print "Setting up manager"
manager.setup(source)

# Set the source
source.set(1,spectrum,ids,qext)
##---------------------------------------------------------------------------##
## SOLVE
##---------------------------------------------------------------------------##

if node() == 0:
    print ">>> Setup complete"
    print ">>> Solving with %s differencing and %s quadrature (%i angles)" \
          % (manager.spatial_descriptor(), angles.quad_label(), angles.num_angles())

if run_problem==1:

    print "Running problem"

    # solve the problem
    manager.solve(angles)

    steady_precursors = Vec_Dbl(mesh.num_cells(),0.0)

    power = Vec_Dbl(mesh.num_cells(),0.0)
    for cell in xrange(mesh.num_cells()):
        matid = mat.matid(cell)
        if mat.assigned_fission(matid):
            for g in xrange(Ng):
                phi = Moments(g)
                power[cell] +=  phi.scalar_flux(cell) \
                        * mat.fission_data(matid, g, NU_SIGMA_F)   \
                        / (1-beta_t)

    power_sum = 0.0
    for cell in xrange(mesh.num_cells()):
        power_sum += power[cell]
    print power_sum

    concentration = ( [], [], [], [], [], [] )

    # Read source term from file and assign to source object
    num_files = 64
    ids = Vec_Dbl(mesh.num_cells(), 0)
    for ifile in xrange(num_files):
        thisfile = open('concentrations'+str(ifile)+'_step2.txt','r')
        firstline = thisfile.readline()
        numcells_file = int(firstline.strip())
        for isrc in xrange(numcells_file*6):
            thisline = thisfile.readline()
            thisline = thisline.strip()
            thisline = thisline.split()
            global_cell = int(thisline[0])
            group    = int(thisline[1])
            conc      = float(thisline[2])
            concentration[group].append(conc)
        thisfile.close()


    for cell in xrange(mesh.num_cells()):
        for i in xrange(6):
            steady_precursors[cell] = steady_precursors[cell] + lamb[i] *
\
                                    (concentration[i][cell] * math.exp(-
lamb[i]*0.01) + \
                                    (1.0/lamb[i]) * beta[i] * power[cell]
\
                                    * (1.0 - math.exp(-lamb[i]*0.01)))

    filename= open('precursors'+str(node())+'_step3.txt','w')
    for cell in xrange(mesh.num_cells()):
        filename.write(str(steady_precursors[cell])+'\n')
    filename.close()

    flux0 = Vec_Dbl(mesh.num_cells(), 0.0)
    flux5 = Vec_Dbl(mesh.num_cells(), 0.0)
    flux10 = Vec_Dbl(mesh.num_cells(), 0.0)
    flux15 = Vec_Dbl(mesh.num_cells(), 0.0)
    flux20 = Vec_Dbl(mesh.num_cells(), 0.0)
    flux25 = Vec_Dbl(mesh.num_cells(), 0.0)
    flux30 = Vec_Dbl(mesh.num_cells(), 0.0)
    flux35 = Vec_Dbl(mesh.num_cells(), 0.0)
    flux40 = Vec_Dbl(mesh.num_cells(), 0.0)

    phi = Moments(0)
    for cell in xrange(mesh.num_cells()):
        flux0[cell]  = phi.scalar_flux(cell)

    phi = Moments(5)
    for cell in xrange(mesh.num_cells()):
        flux5[cell]  = phi.scalar_flux(cell)

    phi = Moments(10)
    for cell in xrange(mesh.num_cells()):
        flux10[cell] = phi.scalar_flux(cell)
    phi = Moments(15)
    for cell in xrange(mesh.num_cells()):
        flux15[cell] = phi.scalar_flux(cell)

    phi = Moments(20)
    for cell in xrange(mesh.num_cells()):
        flux20[cell] = phi.scalar_flux(cell)

    phi = Moments(25)
    for cell in xrange(mesh.num_cells()):
        flux25[cell] = phi.scalar_flux(cell)

    phi = Moments(30)
    for cell in xrange(mesh.num_cells()):
        flux30[cell] = phi.scalar_flux(cell)

    phi = Moments(35)
    for cell in xrange(mesh.num_cells()):
        flux35[cell] = phi.scalar_flux(cell)

    phi = Moments(40)
    for cell in xrange(mesh.num_cells()):
        flux40[cell] = phi.scalar_flux(cell)

    silo = SILO()
    silo.add_mixer(mixer)

    silo.open(str(lat_size)+"x"+str(lat_size)+"_r"+str(Npin)+"a"+str(Nz)+'fixed')

    silo.add("flux0",  flux0)
    silo.add("flux5",  flux5)
    silo.add("flux10", flux10)
    silo.add("flux15", flux15)
    silo.add("flux20", flux20)
    silo.add("flux25", flux25)
    silo.add("flux30", flux30)
    silo.add("flux35", flux35)
    silo.add("flux40", flux40)
    silo.add("power",  power)

    silo.close()

# Write HPCKBA input
if generate_hpckba==1:

    print "Generating HPCKBA input"

    out = HPC_Problem_Output(1, Nx, Ny, Nz)

    print out.chunk(), out.num_chunks_per_file(), out.num_files()

    out.open(str(lat_size)+"x"+str(lat_size)+"_r"+str(Npin)+"a"+str(Nz))

    shapes = Vec_Dbl()

    out.write_db(db)
    out.write_mixed_xs(mixer)
    out.write_src_info(ZERO_SOURCE, ZERO_SOURCE, shapes)

    # write matids
    out.start_field_loop()
    ids = Vec_Int(out.chunk(), 0)
    while not out.finished_field_loop():
        k   = out.current_chunk()
        if k < Nz:
           print "Writing k-plane %d/%d" % (k, Nz)
           for j in xrange(Gy):
               for i in xrange(Gx):
                   cell  = indexer.g2g(i,j,k)
                   index = indexer.g2g(i,j,0)
                   ids[index] = matids[cell]
        out.write_matids(ids)
        out.advance_loop()

    out.close()
##---------------------------------------------------------------------------##
## TIMING
##---------------------------------------------------------------------------##

# output final database (has class-dependent defaults)
db.output()

timer.stop()
time = timer.wall_clock()

keys = timer_keys()
if len(keys) > 0 and node() == 0:
    print "\n"
    print "TIMING : Problem ran in %16.6e seconds." % (time)
    print "------------------------------------------------"
    for key in keys:
        print "%30s : %16.6e" % (key, timer_value(key) / time)
    print "------------------------------------------------"

##---------------------------------------------------------------------------##

manager.close()
finalize()

###############################################################################
## end
###############################################################################
```

VITA

James Ernest Banfield grew up in military family that moved a number of times to various countries and states, including Germany and Japan. James graduated from Lake Braddock Secondary School in Burke, VA in 2004 as a Valedictorian. While in High School, James was awarded an Honorable Mention in the National Teachers Science Association competition of 2002 for work in experimental ultrasound. He then went to the University of Mississippi where he graduated Suma Cum Laude, earning a BS in Mechanical Engineering in 2008. James also graduated from the Sally-Mcdonnel Barksdale Honors College at the University of Mississippi, completing an Undergraduate Thesis on The MagnetoRheological Effect of MagnetoRheological Elastomers. While at the University of Mississippi, James was awarded a variety of scholarships, the most notable of which was the Pichintino Honors College Scholarship. Also as an undergraduate, James worked at Rowan University in a Research Experience for Undergraduates program as well as a design engineering intern at Advanced Distributor Products. Following graduation, James moved to Los Alamos National Laboratory. There, he worked as an intern prior to beginning graduate study in Nuclear Engineering at the University of Tennessee. Once at the University of Tennessee, James was awarded the Department of Energy's Nuclear Engineering University Programs (NEUP) fellowship for three years to pursue a PhD in Nuclear Engineering. During this time, he met his girlfriend Trista Busch as well as adopted a cat, Darius. During his time at Tennessee, James has worked in a continual partnership with Los Alamos, Argonne and Oak Ridge National Laboratories, contributing to a number of technical memos, conference papers, and journals. When not working or volunteering, James enjoys extreme sports such as snowboarding and skateboarding. In skateboarding, James has won two team contests at Woodward Skate Camp in Pennsylvania, where he worked as a counselor in the summer of 2005. James hopes to continue his work with the Department of Energy's National Laboratories, hopefully as a postdoctoral associate or full staff member in the Reactor Physics group of the Reactors and Nuclear Systems Division at ORNL.