Continued Research for Improvement of PROTEUS-SN

Nuclear Engineering Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via DOE's SciTech Connect (<http://www.osti.gov/scitech/>)

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: **orders@ntis.gov**

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: **reports@osti.gov**

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

Continued Research for Improvement of PROTEUS-SN

prepared by
M. A. Smith, V. S. Mahadevan, and E. R. Wolters
Nuclear Engineering Division, Argonne National Laboratory

September 30, 2013

SUMMARY

Over the past several years, the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program of U.S. DOE has focused on developing a neutronics component as part of a larger dynamics product meant to analyze both existing and proposed nuclear reactor systems. The goal is to reduce the uncertainties and biases in reactor design activities by providing enhanced prediction capabilities. As part of the SHARP program, we have invested considerable effort into creating the PROTEUS-SN code. The PROTEUS-SN solver was intended to treat conventional homogenized assembly to explicit geometry schemes for time dependent analysis with coupled thermal-hydraulics and structural mechanics.

For coupled multiphysics time-dependent simulations, a quasi-static kinetics methodology was derived and added to the PROTEUS-SN code. In this method, the point kinetics parameters are updated during the transient using a series of steady-state forward and adjoint spatial calculations. This method would be sufficient to meet the needs of current NEAMS transient analysis in the meantime until fully spatial kinetics methodology is implemented with efficient preconditioned-Krylov solvers. An initial verification test was performed using the two-dimensional numerical benchmark problem in which the control rod was inserted in 0.04 sec and the power was traced up to 100 sec. By comparing the kinetics parameters with the conventional methods, the transient capability of PROTEUS-SN has been verified to be working correctly.

This report also discusses the research work accomplished for improving the PROTEUS-SN code. The specific aspects that we studied include updating the even-parity methodology for which we identified two options: Stabilized Upwind Petrov-Galerkin (SUPG) and Generalized Least Squares (GLS). From the demonstrated benchmark problems, the SUPG and GLS forms showed considerable improvements over the even-parity methodology. This does not mean that SUPG and GLS can unconditionally give better solutions than the discontinuous Galerkin method, but at least the condition number issues we experienced with the even-parity based PROTEUS-SN can be resolved by using SUPG or GLS for fully heterogeneous geometry with strongly varying material cross-sections. We also demonstrate the optimization work required to develop this solver in to a product comparable to existing production codes.

TABLE OF CONTENTS

Summary	i
Table of Contents	ii
List of Figures	iii
List of Tables	iii
1 Introduction	4
2 Updating the PROTEUS-SN Solver for Transient Analysis.....	5
2.1 Kinetics Methodologies	5
2.2 Point Kinetics Method	6
2.3 Quasi-static Kinetics Methodology.....	8
2.4 Implementation	11
2.5 Numerical Examples	12
3 Methodological Work.....	16
3.1 A Test Problem to Verify Spatial Mesh Convergence.....	18
3.2 A Test Problem to Investigate Condition Number Issues.....	19
3.3 Assessment of the SUPG and GLS Operators	22
3.4 Performance Results of the Tetrahedral F matrix	24
4 Conclusions	30
References	31

LIST OF FIGURES

Figure 2.1. Two-Dimensional Fast Reactor Geometry used in Transient Benchmark	12
Figure 2.2. Transient Power Change Over Time	14
Figure 2.3. Initial (Left) and Final (Right) Normalized Power Profiles	14
Figure 2.4. Power Shape at Select Time Points	15
Figure 2.5. Initial (Left) and Final (Right) Power Shape	16
Figure 3.1. Spatial Convergence of Linear Basis Functions	19
Figure 3.2. Spatial Convergence of Cubic Basis Functions	19
Figure 3.3. Checkerboard Problem Domain with Alternating Low (blue) and High (red) Material Cross Section	20
Figure 3.4. Condition Number Variations Depending on $\mu=10$ for All Methods (DG, EP, SUPG, and GLS)	21
Figure 3.5. Condition Number Variations Depending on $\mu=1000$ for All Methods (DG, EP, SUPG, and GLS)	21
Figure 3.6. Matrix Based “Apply F” for a Single Angle and Single Energy Group	24
Figure 3.7. Matrix-free “Apply F” for a Single Angle and Single Energy Group	24
Figure 3.8. Alternate Matrix-free “Apply F” for a Single Angle and Single Energy Group	25
Figure 3.9. Example Coding Instruction and Load/Store/Flop Implementations	29

LIST OF TABLES

Table 2.1. Time Dependent Control Rod Changes and Resulting Kinetics Power	13
Table 3.1. Primary Coefficient Matrix Constant Definitions for Various Methodologies	22
Table 3.2. PAPI Measurements of Matrix Based “Apply F” (-O0)	26
Table 3.3. PAPI Measurements on Matrix-free “Apply F” (-O0)	26
Table 3.4. PAPI Measurements on Alternate Matrix-free “Apply F” (-O0)	26
Table 3.5. Summary of PAPI Measurements of -O0 for “Apply F”.	26
Table 3.6. PAPI Measurements of Matrix Based “Apply F” (-O3)	27
Table 3.7. PAPI Measurements on Matrix-free “Apply F” (-O3)	27
Table 3.8. PAPI Measurements on Alternate Matrix-free “Apply F” (-O3)	27
Table 3.9. Summary of PAPI Measurements of -O3 for “Apply F”.	27
Table 3.10. GFlops/s Estimate Using the Simplistic L_1 Cache Performance Model	30

1 Introduction

Under the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program of U.S. DOE, the SHARP team at ANL has continued the development of a suite of modern simulation tools for use in all reactor types of interest. The general goal is to reduce the uncertainties and biases in various areas of reactor design activities by providing enhanced prediction capabilities. As a fast reactor component of SHARP, a high-fidelity deterministic neutron transport code named PROTEUS-SN [1-6] was developed specially for analysis and design of sodium fast reactors (SFRs). The application scope of PROTEUS-SN was to range from conventional homogenized assembly approaches to explicit geometry, time-dependent transport calculations coupled with thermal-hydraulics and structural mechanics for reactor accident simulations.

To support the time-dependent multi-physics modeling capability that is one of the goals of NEAMS, we added a time-dependent modeling capability to PROTEUS-SN. Because the current production version of PROTEUS-SN does not have a top level Krylov (GMRES) algorithm, we could not realistically implement any of the more rigorous fully-implicit kinetics schemes and thus rely on quasi-static kinetics approximations. In this methodology, the point kinetics parameters are updated at each time step via a steady-state forward and adjoint flux calculation. It should be noted that a quasi-static kinetics approach should be sufficient to meet the needs of current NEAMS transient analysis until a more rigorous kinetics methodology is implemented which requires an efficient GMRES solution algorithm.

The version of PROTEUS-SN that SHARP has relied upon was initially developed in 2008-2009 and was rapidly built to prove that we could fully utilize a supercomputer (>200,000 cores) which had not been demonstrated before. PROTEUS-SN was able to achieve great scaling on large unstructured mesh problem sizes whose, to date, peers have been a few structured grid solvers. To accomplish this feat, a simplistic solution algorithm was chosen such that the demonstration product could be built rapidly. As a consequence, the code may not be well posed to solve problems with $>10^9$ elements and has been observed to run slowly on thermal reactor problems. Note that the simplistic algorithm was the primary reason that we initially implemented a quasi-static kinetics approach to PROTEUS-SN for its kinetics capability, even though the approach is not the best choice for heterogeneous geometries.

In this report we also demonstrate the research work required to complete modifications to PROTEUS-SN and make it the production version that demonstrates improved performance on transport calculations for heterogeneous geometry problems. The specific aspects that we studied include analysis on other continuous Galerkin based formulations for the first-order neutron transport equation: Stabilized Upwind Petrov-Galerkin (SUPG) and Generalized Least Squares (GLS) methods. While it is possible to replace the even-parity methodology in PROTEUS-SN with a SUPG or GLS scheme, this does not address the fundamental problem of memory usage and poor computing efficiency that prevents PROTEUS-SN from handling the large dynamics problems targeted by NEAMS. To address this issue, we provide a demonstration of the research necessary to build, optimize and improve the PROTEUS-SN solver.

In the following section we discuss the updates to the PROTEUS-SN code to allow transient modeling. From there we discuss the work done to address the methodological problems with the existing PROTEUS-SN solver and allude to the actual requirements necessary to provide PROTEUS-SN with a production capability.

2 Updating the PROTEUS-SN Solver for Transient Analysis

Given the restrictions placed upon this work by the status of the existing PROTEUS-SN code and the desire to enable some transient modeling within the NEAMS activity, the quasi-static kinetics capability [7] was added to the PROTEUS-SN solver [1-3] as a balance between cost and accuracy. This is of course the bare minimum kinetics capability where improved quasi-static [8-9] is the typically accepted scheme followed by full discretizations of the time dependent transport equation [9]. Because the version of PROTEUS-SN that we are using does not have a top level GMRES algorithm, we cannot realistically implement any of the more rigorous schemes and thus rely upon the basic kinetics algorithm. The remainder of this section focuses on the derivation of the kinetics methodology, its implementation into PROTEUS-SN, and example results.

2.1 Kinetics Methodologies

Any kinetics derivation starts with the time dependent transport equation

$$\frac{1}{v} \frac{\partial}{\partial t} \psi(\vec{r}, E, \hat{\Omega}, t) + \nabla \cdot \hat{\Omega} \psi(\vec{r}, E, \hat{\Omega}, t) + \Sigma_t(\vec{r}, E, t) \psi(\vec{r}, E, \hat{\Omega}, t) = S(\vec{r}, E, \hat{\Omega}, t), \quad (2.1)$$

and the composite source

$$S(\vec{r}, E, \hat{\Omega}, t) = W(\vec{r}, E, \hat{\Omega}, t) + F(\vec{r}, E, \hat{\Omega}, t), \quad (2.2)$$

which is separated into scattering, $W(\vec{r}, E, \hat{\Omega}, t)$, and fission sources $F(\vec{r}, E, \hat{\Omega}, t)$. The fission source requires a special time dependent treatment because of the time required for an excited state nucleus to decay and release neutrons. The typical modeling approach is to assume a 0th order finite element which breaks the continuum of delay neutrons into a few “families” each of which has a time decay constant, λ_m and generation probability ν_m . The remaining “prompt” or instantaneous neutrons are denoted using ν_p . The fission source in this case is written as the system of coupled equations where i denotes each isotope at a given position in space and m denotes each family.

$$F(\vec{r}, E, \hat{\Omega}, t) = \sum_i N_i(\vec{r}, t) \chi_{p,i}(\vec{r}, E, t) \int dE' \nu_{p,i} \sigma_{f,i}(\vec{r}, E', t) \int d\Omega' \psi(\vec{r}, E', \hat{\Omega}', t) \\ + \sum_i \sum_{m=1}^M \lambda_{m,i} C_{m,i}(\vec{r}, E, t) \quad (2.3)$$

$$\frac{\partial C_{m,i}(\vec{r}, E, t)}{\partial t} = N_i \int dE' \chi_{i,m} \sigma_{f,i}(\vec{r}, E, E', t) \int d\Omega' \psi(\vec{r}, E', \hat{\Omega}', t) - \lambda_m(\vec{r}) C_{m,i}(\vec{r}, E, t) \quad (2.4)$$

Clearly from the above system, implementation of a standard kinetics algorithm will add a considerable number of degrees of freedom to the problem being solved (20 actinides

with 6 delay families is 120 degrees of freedom per mesh that contains fuel noting that the precursors are a function of energy).

2.2 Point Kinetics Method

In point kinetics, we eliminate the space and isotopic dependence by computing “core” averaged quantities. The transport equation is also truncated such that “linear reactivity effects” are added such as sodium density coefficient of reactivity. The point kinetics equation begins assuming a separation of the time and space-angle-energy dependence of the flux and precursor concentrations. This means that a “steady state” shape exists which is the solution to the time independent equation:

$$\nabla \cdot \hat{\Omega} \psi_0(\vec{r}, E, \hat{\Omega}) + \Sigma_t(\vec{r}, E) \psi_0(\vec{r}, E, \hat{\Omega}) = W_0(\vec{r}, E, \hat{\Omega}) + \frac{1}{k_0} F_0(\vec{r}, E, \hat{\Omega}), \quad (2.5)$$

$$F_0(\vec{r}, E, \hat{\Omega}) = \sum_i N_i(\vec{r}) \chi_{T,i}(\vec{r}, E) \int dE' \nu_{T,i} \sigma_{f,i}(\vec{r}, E') \int d\Omega' \psi_0(\vec{r}, E', \hat{\Omega}').$$

The initial precursor concentrations are found to be

$$C_{m,i,0}(\vec{r}, E, t) = \frac{N_i \int dE' \chi_{i,m} \sigma_{f,i}(\vec{r}, E, E') \int d\Omega' \psi_0(\vec{r}, E', \hat{\Omega}')}{\lambda_m(\vec{r})}. \quad (2.6)$$

Note that equation 2.5 uses the total $\nu_{T,i}$ corresponding to substituting equation 2.6 into equation 2.3. With this, we can assume the flux and precursor concentrations can be written as

$$\psi(\vec{r}, E, \hat{\Omega}, t) \rightarrow n(t) \psi_0(\vec{r}, E, \hat{\Omega}) \quad (2.7)$$

$$C_{m,i}(\vec{r}, E, t) \rightarrow b(t) C_{m,i,0}(\vec{r}, E, t) \rightarrow c_{m,i}(t) \psi_0(\vec{r}, E, \hat{\Omega}). \quad (2.8)$$

Equation 2.7 is valid so long as the time dependent constant is taken to be the operator (delay fission source divided by decay constant) appearing in equation 2.6. Plugging these into equations 2.1 through 2.3 yields

$$\frac{1}{\nu} \frac{\partial}{\partial t} n(t) \psi_0(\vec{r}, E, \hat{\Omega}) = \sum_i \sum_{m=1}^M \lambda_{m,i} c_{m,i}(t) \psi_0(\vec{r}, E, \hat{\Omega}) - n(t) \left\{ \nabla \cdot \hat{\Omega} \psi_0(\vec{r}, E, \hat{\Omega}) + \Sigma_t(\vec{r}, E, t) \psi_0(\vec{r}, E, \hat{\Omega}) - W_0(\vec{r}, E, \hat{\Omega}) - F_p(\vec{r}, E, \hat{\Omega}) \right\}. \quad (2.9)$$

$$F_p(\vec{r}, E, \hat{\Omega}) = \sum_i N_i(\vec{r}) \chi_{p,i}(\vec{r}, E) \int dE' \nu_{p,i} \sigma_{f,i}(\vec{r}, E') \int d\Omega' \psi_0(\vec{r}, E', \hat{\Omega}')$$

Note that the steady state transport equation is nearly present as a multiplier on $n(t)$ except that the k_0 eigenvalue concept is not present and the fission source only includes the prompt contribution. By adding the missing component one can collapse the equation to

$$\frac{1}{\nu} \frac{\partial}{\partial t} n(t) \psi_0(\vec{r}, E, \hat{\Omega}) = \sum_i \sum_{m=1}^M \lambda_{m,i} c_{m,i}(t) \psi_0(\vec{r}, E, \hat{\Omega}) - n(t) \left\{ F_p(\vec{r}, E, \hat{\Omega}) + \frac{1}{k_0} F_0(\vec{r}, E, \hat{\Omega}) \right\}. \quad (2.10)$$

The typical approach in point kinetics is to eliminate the space-angle-energy dependence of the flux equation in favor of a one-group “homogeneous” diffusive system. This allows one to utilize the equations

$$\frac{\partial n(t)}{\partial t} = \frac{p(t) - \beta}{\Lambda} n(t) + \sum_{m=1}^M \lambda_m c_m(t) \quad (2.11)$$

$$\frac{\partial c_m(t)}{\partial t} = \frac{\beta_m}{\Lambda} n(t) - \lambda_m c_m(t) \quad (2.12)$$

$$p(t) = \frac{1}{k_0} - \frac{1}{k(t)}. \quad (2.13)$$

In this case, we can compute initial conditions for $c_m(0)$, but not for $n(0)$, noting that $p(0) = 0$ for any reactor system via:

$$c_m(0) = \frac{\beta_m}{\Lambda \cdot \lambda_m} n(0) \quad (2.14)$$

$$0 = \frac{p(0) - \beta}{\Lambda} n(0) + \sum_{m=1}^M \lambda_m \frac{\beta_m}{\Lambda \cdot \lambda_m} n(0). \quad (2.15)$$

To obtain the initial condition for $n(0)$, one can effectively assume anything (say 2.3), but it should be scalable to the power of the system in question. In this case, we assume a solution of equation 2.5 exists with a stated power such that $n(0) = 1$. At any time step, we can then use equation 2.7 to renormalize the magnitude of $\psi_0(\vec{r}, E, \hat{\Omega})$.

There are many ways to discretized equations 2.11 and 2.12 and we show only the most basic one here which does exist as one option in PROTEUS-SN. We can integrate equations 2.11 and 2.12 over the time interval t_j to t_{j+1} using a fully implicit formulation to get

$$\frac{n_{j+1} - n_j}{t_{j+1} - t_j} = \frac{p_{j+1} - \beta}{\Lambda} n_{j+1} + \sum_{m=1}^M \lambda_m c_{m,j+1} \quad (2.16)$$

$$\frac{c_{m,j+1} - c_{m,j}}{t_{j+1} - t_j} = \frac{\beta_m}{\Lambda} n_{j+1} - \lambda_m c_{m,j+1} \quad (2.17)$$

Assuming a linear change in $p(t)$ over a time step we can write

$$p_{j+1} = \frac{1}{k_j} - \frac{1}{k_{j+1}}. \quad (2.18)$$

Rearranging terms, we obtain

$$\left\{ \frac{1}{t_{j+1} - t_j} - \frac{p_{j+1} - \beta}{\Lambda} \right\} n_{j+1} + \sum_{m=1}^M \{-\lambda_m\} c_{m,j+1} = \left\{ \frac{1}{t_{j+1} - t_j} \right\} n_j \quad (2.19)$$

$$\left\{ \frac{1}{t_{j+1} - t_j} + \lambda_m \right\} c_{m,j+1} + \left\{ \frac{-\beta_m}{\Lambda} \right\} n_{j+1} = \left\{ \frac{1}{t_{j+1} - t_j} \right\} c_{m,j} \quad (2.20)$$

This system is obviously dependent upon the time step size $(\Delta t_{j+1} = t_{j+1} - t_j)$ and the computed k_{j+1} at the end of the time step. These equations are rather easy to implement in a linear algebra solver and propagate for a given time step size.

In a conventional point kinetics system, the reactivity appearing in equations 2.11 and 2.12 is written as

$$\rho(t) = f_{\text{temperature}} \cdot T + f_{\text{density}} \cdot \rho + \dots f_* \cdot F_*. \quad (2.21)$$

where the reactivity coefficients f_* (such as sodium density or fuel temperature) and its associated dependent variable F_* are typically coarse grained quantities obtained from the other physics areas. Note that each of these terms can be broken down into a sum of contributions from each mesh in the domain, but that under no circumstance is the neutronics calculation recomputed during the transient modeling.

2.3 Quasi-static Kinetics Methodology

The basic difference between quasi-steady state kinetics implementations and point kinetics is that the reactivity equation given by equation 2.21 is replaced with a series of steady state neutronics calculations. In conventional quasi-steady state kinetics, a separation of the time and space-angle-energy dependence is assumed for the flux and precursor concentrations to get

$$\psi(\vec{r}, E, \hat{\Omega}, t) \rightarrow n(t)\psi(\vec{r}, E, \hat{\Omega}, t) \quad (2.22)$$

$$C_{m,i}(\vec{r}, E, t) \rightarrow b(t)C_{m,i,0}(\vec{r}, E, t) \rightarrow c_{m,i}(t)\psi(\vec{r}, E, \hat{\Omega}, t). \quad (2.23)$$

The primary goal is to isolate the space shape of the flux from the time dependence. In this manner, we take a solution to the steady state neutron transport equation of

$$T(\vec{r}, E, \hat{\Omega})\psi(\vec{r}, E, \hat{\Omega}) = W(\vec{r}, E, \hat{\Omega}) + \frac{1}{k}F(\vec{r}, E, \hat{\Omega}) \quad (2.24)$$

where the operators are defined to be

$$T(\vec{r}, E, \hat{\Omega}) = \nabla \cdot \hat{\Omega} + \Sigma_t(\vec{r}, E) \quad (2.25)$$

$$F(\vec{r}, E, \hat{\Omega}) = \sum_i N_i(\vec{r}) \chi_{T,i}(\vec{r}, E) \int dE' \nu_{T,i} \sigma_{f,i}(\vec{r}, E') \int d\Omega' \psi(\vec{r}, E', \hat{\Omega}') \quad (2.26)$$

$$W(\vec{r}, E, \hat{\Omega}) = \int dE' \int d\Omega' \sigma_s(\vec{r}, \hat{\Omega}, \hat{\Omega}', E, E') \psi(\vec{r}, E', \hat{\Omega}'). \quad (2.27)$$

We can also obtain the adjoint flux solution $\psi^*(\vec{r}, E, \hat{\Omega})$ for the transport equation

$$T^*(\vec{r}, E, \hat{\Omega})\psi^*(\vec{r}, E, \hat{\Omega}) = W^*(\vec{r}, E, \hat{\Omega}) + \frac{1}{k}F^*(\vec{r}, E, \hat{\Omega}), \quad (2.28)$$

noting that the eigenvalue between the forward (equation 2.24) and adjoint equations is identical.

We write equations 2.24 and 2.28 in discrete forms thus yielding the matrix vector systems:

$$(\underline{T} - \underline{W})\underline{\psi} = \underline{A}\underline{\psi} = \frac{1}{k} \underline{F}\underline{\psi} \quad (2.29)$$

$$(\underline{T}^* - \underline{W}^*)\underline{\psi}^* = \underline{A}^*\underline{\psi}^* = \frac{1}{k} \underline{F}^*\underline{\psi}^*. \quad (2.30)$$

Looking back at equations 2.22 and 2.23, we can assume they infer

$$\underline{\psi}(\vec{r}, E, \hat{\Omega}, t) \rightarrow n(t)\underline{\psi}(\vec{r}, E, \hat{\Omega}, t) = n(t)\underline{\psi}(t) \quad (2.31)$$

$$C_{m,i}(\vec{r}, E, t) \rightarrow c_{m,i}(t)\underline{\psi}(\vec{r}, E, \hat{\Omega}, t) = c_{m,i}(t)\underline{\psi}(t), \quad (2.32)$$

we can write the time dependent transport and precursor equations as:

$$\left\{ \frac{1}{v} \frac{\partial}{\partial t} + \underline{A}(t) - \underline{F}_p(t) \right\} n(t)\underline{\psi}(t) = \sum_{m,i} \lambda_{m,i} c_{m,i}(t)\underline{\psi}(t) \quad (2.33)$$

$$\frac{\partial}{\partial t} c_{m,i}(t)\underline{\psi}(t) = \underline{F}_{m,i,d}(t)n(t)\underline{\psi}(t) - \lambda_{m,i} c_{m,i}(t)\underline{\psi}(t). \quad (2.34)$$

At this point, the form is exact as the operators change with respect to time.

Focusing on equation 2.33 first, we expand the derivative to write

$$\frac{1}{v} \underline{\psi}(t) \frac{\partial n(t)}{\partial t} + \frac{1}{v} n(t) \frac{\partial \underline{\psi}(t)}{\partial t} + \{ \underline{A}(t) - \underline{F}_p(t) \} n(t)\underline{\psi}(t) = \sum_{m,i} \lambda_{m,i} c_{m,i}(t)\underline{\psi}(t). \quad (2.35)$$

We can similarly expand equation 2.34 to obtain

$$\underline{\psi}(t) \frac{\partial c_{m,i}(t)}{\partial t} + c_{m,i}(t) \frac{\partial \underline{\psi}(t)}{\partial t} = \underline{F}_{m,i,d}(t)n(t)\underline{\psi}(t) - \lambda_{m,i} c_{m,i}(t)\underline{\psi}(t). \quad (2.36)$$

In the next step, we assume the time derivative of the shape function over a given time step is negligible such that

$$\frac{\partial \underline{\psi}(t)}{\partial t} \approx 0. \quad (2.37)$$

This simplifies equations 2.14 and 2.15 to

$$\{ \underline{A}(t) - \underline{F}_p(t) \} n(t)\underline{\psi}(t) = \sum_{m,i} \lambda_{m,i} c_{m,i}(t)\underline{\psi}(t) - \frac{1}{v} \underline{\psi}(t) \frac{\partial n(t)}{\partial t}. \quad (2.38)$$

$$\underline{\psi}(t) \frac{\partial c_{m,i}(t)}{\partial t} = \underline{F}_{m,i,d}(t)n(t)\underline{\psi}(t) - \lambda_{m,i} c_{m,i}(t)\underline{\psi}(t). \quad (2.39)$$

To initialize the system, we assume there are an initial forward and adjoint flux solutions $\underline{\psi}_0$ and $\underline{\psi}_0^*$. We weight equations 2.38 and 2.39 with the adjoint flux to get:

$$\frac{\underline{\psi}_0^{*T} \underline{A}(t)\underline{\psi}(t) - \underline{\psi}_0^{*T} \underline{F}_p(t)\underline{\psi}(t)}{\underline{\psi}_0^{*T} \underline{\psi}(t)} n(t) = \sum_{m,i} \lambda_{m,i} c_{m,i}(t) - \frac{\underline{\psi}_0^{*T} \frac{1}{v} \underline{\psi}(t)}{\underline{\psi}_0^{*T} \underline{\psi}(t)} \frac{\partial n(t)}{\partial t}. \quad (2.40)$$

$$\frac{\partial c_{m,i}(t)}{\partial t} = \frac{\underline{\psi}_0^{*T} \underline{F}_{m,i,d}(t)\underline{\psi}(t)}{\underline{\psi}_0^{*T} \cdot \underline{\psi}(t)} n(t) - \lambda_{m,i} c_{m,i}(t). \quad (2.41)$$

Writing the point kinetics equations with isotopic precursor concentrations we get

$$\frac{\partial n(t)}{\partial t} = \frac{\rho(t) - \beta}{\Lambda} n(t) + \sum_{m,i} \lambda_{m,i} c_{m,i}(t) \quad (2.42)$$

$$\frac{\partial c_{m,i}(t)}{\partial t} = \frac{\beta_{m,i}}{\Lambda} n(t) - \lambda_{m,i} c_{m,i}(t). \quad (2.43)$$

which are basically the same as equations 2.11 and 2.12. Comparing equations 2.42 and 2.43 to equations 2.40 and 2.41, we find the following definitions make the two systems of equations equivalent:

$$\frac{\psi_0^{*T} \frac{1}{v} \psi(t)}{\psi_0^{*T} \psi(t)} \equiv 1 \quad (2.44)$$

$$\frac{p(t) - \beta}{\Lambda} = - \frac{\psi_0^{*T} \underline{A}(t) \psi(t) - \psi_0^{*T} \underline{F}_p(t) \psi(t)}{\psi_0^{*T} \psi(t)} \quad (2.45)$$

$$\frac{\beta_{m,i}}{\Lambda} = \frac{\psi_0^{*T} \underline{F}_{m,i,d}(t) \psi(t)}{\psi_0^{*T} \psi(t)}. \quad (2.46)$$

The preference is to eliminate the operator usage in equation 2.45 in favor of the simple reactivity form of equation 2.45, thus we expend some effort here to show the transformation. First, we note the constraint imposed by 44 and assume we will see a normalization factor associated with $\psi(t) = f(t) \cdot \hat{\psi}(t)$ such that $f^{-1}(t) = \frac{\psi_0^{*T} \frac{1}{v} \psi(t)}{\psi_0^{*T} \psi(t)}$.

Focusing just on the operator terms, we know the initial condition to be given as

$$\psi_0^{*T} \underline{A}(0) \psi_0 = \frac{1}{k_0} \psi_0^{*T} \underline{F}(0) \psi_0. \quad (2.47)$$

We can assume the perturbed operator has a steady state flux solution defined by

$$\psi_0^{*T} \underline{A}(t) \psi(t) = \frac{1}{k(t)} \psi_0^{*T} \underline{F}(t) \psi(t). \quad (2.48)$$

Modifying equation 2.24 to be in this form, we can write

$$\frac{p(t) - \beta}{\Lambda} = - \left\{ \frac{\psi_0^{*T} \underline{A}(t) \psi(t)}{\psi_0^{*T} \underline{F}(t) \psi(t)} - \frac{\psi_0^{*T} \underline{F}_p(t) \psi(t)}{\psi_0^{*T} \underline{F}(t) \psi(t)} \right\} \frac{\psi_0^{*T} \underline{F}(t) \psi(t)}{\psi_0^{*T} \psi(t)}. \quad (2.49)$$

Taking note of the point kinetics definitions for Λ and $\beta_{m,i}$ given as

$$\Lambda = \frac{\psi_0^{*T} \frac{1}{v} \psi_0}{\psi_0^{*T} \underline{F} \psi_0} \quad \beta_{m,i} = \frac{\psi_0^{*T} \underline{F}_{m,i,d} \psi_0}{\psi_0^{*T} \underline{F} \psi_0}, \quad (2.50)$$

and the constraint imposed by equation 2.23, we can further modify equation 2.49 as

$$\begin{aligned} \frac{p(t) - \beta}{\Lambda} &= \left\{ 1 - \frac{1}{k(t)} + \frac{\sum_{m,i} \psi_0^{*T} \underline{F}_{m,i,d}(t) \hat{\psi}(t)}{\psi_0^{*T} \underline{F}(t) \hat{\psi}(t)} \right\} \frac{1}{\Lambda} \frac{\psi_0^{*T} \underline{F}(t) \hat{\psi}(t)}{\psi_0^{*T} \underline{F} \psi_0} \frac{\psi_0^{*T} \frac{1}{v} \psi_0}{\psi_0^{*T} \hat{\psi}(t)} \\ &= \left\{ 1 - \frac{1}{k(t)} - \beta(t) \right\} \frac{1}{\Lambda} g(t) \end{aligned} \quad (2.51)$$

Clearly equation 2.51 can be made equivalent if the initial eigenvalue is 1 and the fission source remains unchanged over the given time step. Further manipulation to incorporate the exact form is difficult as the underlying $\beta_{m,i}$ can change with time due to spectral changes in

the cross section data (or more unlikely, changes in fissionable material content). Focusing on equation 2.46, we can introduce the scaling factor and get

$$\frac{\beta_{m,i}}{\Lambda} = \frac{1}{\Lambda} \frac{\psi_0^{*T} \underline{F}_{m,i,d}(t) \hat{\psi}(t)}{\psi_0^{*T} \underline{F}(t) \hat{\psi}(t)} \frac{\psi_0^{*T} \underline{F}(t) \hat{\psi}(t)}{\psi_0^{*T} \underline{F} \psi_0} \frac{\psi_0^{*T} \frac{1}{v} \psi_0}{\psi_0^{*T} \cdot \hat{\psi}(t)} = \frac{\beta_{m,i}(t)}{\Lambda} g(t). \quad (2.52)$$

We note this does not change the fact that the inner products still have to be computed.

2.4 Implementation

Equations 2.42 and 2.43 form the basis of the kinetics methodology implemented into the PROTEUS-SN code although we note that one could make the kinetics parameters β and Λ functions of time. Focusing on the basic implementation, we have initial conditions:

$$\underline{A}_0 \psi_0 = \frac{1}{k_0} \underline{F}_0 \psi_0 \quad \underline{A}_0^* \psi_0^* = \frac{1}{k_0} \underline{F}_0^* \psi_0^* \quad \Lambda_0 = \frac{\psi_0^{*T} \frac{1}{v} \psi_0}{\psi_0^{*T} \underline{F}_0 \psi_0} \quad \beta_{m,i,0} = \frac{\psi_0^{*T} \underline{F}_{m,i,d,0} \psi_0}{\psi_0^{*T} \underline{F}_0 \psi_0} \quad (2.53)$$

With the solutions to these equations and the power level (n_0), we can obtain the initial precursor concentrations

$$c_{m,0} = \frac{\beta_{m,0}}{\Lambda_0 \cdot \lambda_m} n_0. \quad (2.54)$$

At the end of each time step, we obtain a steady state flux solution to the new problem configuration

$$\underline{A}_t \psi_t = \frac{1}{k_t} \underline{F}_t \psi_t. \quad (2.55)$$

The eigenvalue k_t is used to define the reactivity

$$\rho_t = \frac{1}{k_{t-1}} - \frac{1}{k_t} \quad (2.56)$$

which in turn is used in the point kinetics equations reiterated as:

$$\frac{\partial n(t)}{\partial t} = \frac{\rho_t - \beta_0}{\Lambda_0} n(t) + \sum_{m,i} \lambda_{m,i} c_{m,i}(t) \quad (2.57)$$

$$\frac{\partial c_{m,i}(t)}{\partial t} = \frac{\beta_{m,i,0}}{\Lambda_0} n(t) - \lambda_{m,i} c_{m,i}(t). \quad (2.58)$$

From here, we have demonstrated the necessary steps to discretize and solve equations 2.57 and 2.58 noting that we use the option defined earlier in equations 2.16 through 2.20. As one would expect, one could, and in some circumstances should, update the kinetics parameters appearing in equations 2.57 and 2.58 with the progression in time. As an example, the movement of fuel out of the core or massive removal of coolant will change these parameters significantly relative to their initial conditions. However, given the relatively inaccurate approach that quasi-static kinetics poses, it is safe to assume that the changes in the kinetics parameters will be minor such that one only needs to update them infrequently.

In the PROTEUS-SN code, the kinetics input option “UPDATE_LAMBDABETA NO” specifies the solution of equations 2.53 through 2.58 while “UPDATE_LAMBDABETA YES” will alter the equations to the form:

$$\frac{\partial n(t)}{\partial t} = \frac{p_t - \beta_t}{\Lambda_t} n(t) + \sum_{m,i} \lambda_{m,i} c_{m,i}(t) \quad (2.59)$$

$$\frac{\partial c_{m,i}(t)}{\partial t} = \frac{\beta_{m,i,t}}{\Lambda_t} n(t) - \lambda_{m,i} c_{m,i}(t), \quad (2.60)$$

where the kinetics parameters are updated at the end of each time step by additionally solving the equations

$$\underline{A}_t^* \underline{\psi}_t^* = \frac{1}{k_t} \underline{F}_t^* \underline{\psi}_t^* \quad \Lambda_t = \frac{\underline{\psi}_t^{*T} \underline{F}_t^* \underline{\psi}_t^*}{\underline{\psi}_t^{*T} \underline{F}_t^* \underline{\psi}_t^*} \quad \beta_{m,i,t} = \frac{\underline{\psi}_t^{*T} \underline{F}_{m,i,d,t} \underline{\psi}_t^*}{\underline{\psi}_t^{*T} \underline{F}_t^* \underline{\psi}_t^*}. \quad (2.61)$$

It is not clear how reliable this scheme is and we suggest considerable caution when using either kinetics options in PROTEUS-SN.

2.5 Numerical Examples

While several verification test problems were created, we only present one here for brevity and note that we were unable to include the new version into the coupled physics milestone due at the end of the year. Figure 2.1 shows the geometry definition for the 2D benchmark we used to verify the PROTEUS-SN transient capability where the bottom and left boundary conditions are reflected and the top and right boundary conditions are void.

Figure 2.1. Two-Dimensional Fast Reactor Geometry used in Transient Benchmark

One can see that Figure 2.1 is a rather simplistic reactor geometry and that the part of interest for this work is the change in the composition of the control rod. The benchmark problem came with 3 group fast spectrum cross section data and 2 delay families rather than the typical 33 group cross section data and 6 delay families. The details of the cross section data are neglected for brevity. Table 2.1 shows the transient setup for the kinetics problem associated with Figure 2.1 along with the steady state eigenvalue solutions and computed power. Figure 2.2 plots the detailed power distribution over the 100 second time frame where the short term effects occurring in the first 0.04 seconds are given on the left and the long term transient change is given on the right.

Table 2.1. Time Dependent Control Rod Changes and Resulting Kinetics Power

Time (seconds)	Fractional Control Rod Insertion	Steady State Eigenvalues	Computed Power (Watts)
0.0	0.0%		100.0
0.0005	2.5%	1.17714	95.1
0.0010			85.3
0.0100			84.3
0.0105	5.0%	1.17638	80.7
0.0110			74.1
0.0200			73.5
0.0205	7.5%	1.17562	70.6
0.0210			65.5
0.0300			65.1
0.0305	10.0%	1.17493	62.9
0.0310			59.1
0.0400			58.9
0.0405	12.5%	1.17422	57.0
0.0410			53.9
0.0500			53.7
0.0750			53.4
0.1000			53.1
0.2000			52.3
0.5000			49.8
1.0000			45.9
2.0000			40.7
5.0000			34.4
10.000			30.0
100.000			13.3

Figure 2.2. Transient Power Change Over Time

As can be seen in Figure 2.2, the ~ 300 pcm change in the eigenvalue spread out over five time steps has the characteristic prompt neutron drop followed by the gradual decay. This result is exactly the expected result from theory and we note that increasing or decreasing the control rod “insertion” alters the power as one would expect. This benchmark was chosen because it could easily be verified using the PERSENT perturbation theory code [10-16] at each steady state configuration. The initial Λ and β from PERSENT were computed to be $5.03 \cdot 10^{-7}$ and 0.00325 while PROTEUS-SN found them to be $5.07 \cdot 10^{-7}$ and 0.00324. The difference is believed to be a consequence of the space-angle refinement and was not investigated further in this work. The same quantities at the end of the calculation are $5.04 \cdot 10^{-7}$ and 0.00324 which is not unusual given the relatively small material change made to the system.

Figure 2.3. Initial (Left) and Final (Right) Normalized Power Profiles

Figure 2.4. Power Shape at Select Time Points

Figure 2.3 plots the normalized radial power profiles from the beginning and end of the calculation to highlight the impact of the asymmetric control rod “insertion.” Figure 2.4 shows selected plots of the power distribution over the first 10 seconds combining the radial shape with the reduction in power seen in Figure 2.3. Finally, Figure 2.5 shows the overall change in the power from the initial condition to the final condition. Starting with Figure 2.4, one can clearly see the impact of the control rod “insertion” as a localized alteration of the flux profile near the modified control rod position. Looking at Figure 2.4, clearly the power drop dominates the radial shape changes in the flux magnitude over the time scale and thus mostly obscures the ability to see the build in of the radial shape change. However, looking at time step 0.0105 seconds, one can clearly see that an asymmetry in the flux solution near the control rod indicating that the radial shape change is almost immediate. With regard to Figure 2.5 we only included it for completeness and to denote the fact that PROTEUS-SN transient capability can be used for short term and long term transient modeling.

In general, the actual shape changes in the flux distribution are hard to discern on kinetics problems due to the relatively small magnitude insertion being studied. In this case, a 300 pcm insertion of reactivity is nearly prompt sub-critical. For problems with rather minor changes in the radial profile such as this one, we can expect the quasi-static kinetics scheme to be very accurate even with its known limitations. No additional effort was taken to investigate the accuracy of this calculation relative to other kinetics codes or a known reference solution.

Figure 2.5. Initial (Left) and Final (Right) Power Shape

3 Methodological Work

The most concerning part of the preceding PROTEUS-SN transient work is that there was insufficient time to verify or validate that it was the appropriate approach to follow. There are considerable issues with deterministic transport codes associated with transitioning from an assembly homogenized methodology to a fully heterogeneous one and making assumptions about the dynamics modeling capability will have serious consequences with regard to solution accuracy. The consistent problems we have had with defining cross sections for these heterogeneous calculations over the past three years along with the inadequacies of

the quasi-static kinetics and even-parity discrete ordinate methodology make it hard to promote the PROTEUS-SN development within the broader neutronics community.

In an attempt to fix one of these issues, we have started research on alternative methodologies that resolve the condition number issues which plague the even-parity method. To date, we have spent time researching scalable multigrid preconditioners for FEM discretizations of the even-parity transport equation [18]. From our experiences, the second-order forms (even-parity or self-adjoint angular flux) become ill-posed or severely ill-conditioned when applied to strongly heterogeneous media with fine spatial meshes along with any problem containing low density (void) regions. To counter these problems we investigated a family of Least Squares (LS), Generalized Least Squares (GLS), and stabilized continuous Galerkin (SUPG) FEM formulations. The motivation of course is to take advantage of the past research on the even-parity transport equation that was demonstrated to be well posed for petascale and beyond architectures.

To understand the basics of the SUPG continuous FEM discretization, we will only consider the within group neutron transport equation, since the multi-group energy and discrete ordinates (S_N) based angular discretizations yield a coupled set of similar discrete equations.

$$\nabla \cdot \hat{W} \mathcal{Y}(\vec{r}, \hat{W}) + S_t(\vec{r}) \mathcal{Y}(\vec{r}, \hat{W}) = S(\vec{r}, \hat{W}), \quad (3.1)$$

where the composite source

$$S(\vec{r}, \hat{W}) = W(\vec{r}, \hat{W}) + F(\vec{r}, \hat{W}) = \int dW S_s(\vec{r}, \hat{W}' \cdot \hat{W}) \mathcal{Y}(\vec{r}, \hat{W}') + F(\vec{r}, \hat{W}), \quad (3.2)$$

can be considered to consist of scattering, $W(\vec{r}, \hat{W})$, and a fixed source $F(\vec{r}, \hat{W})$. With S_N angular discretization, this can be considered a set of coupled equations of the form

$$R_n(\vec{r}) = \hat{W}_n \cdot \nabla \mathcal{Y}_n(\vec{r}) + S_t(\vec{r}) \mathcal{Y}_n(\vec{r}) - S_n(\vec{r}) = 0. \quad (3.3)$$

In this context, the weak form of Equation 3.1 can be written as:

$$A(\mathcal{Y}_n(\vec{r}), v_n(\vec{r})) = B(\mathcal{Y}_n(\vec{r}), v_n(\vec{r})) + L(v_n(\vec{r})). \quad (3.4)$$

where

$$\begin{aligned} A(\mathcal{Y}_n(\vec{r}), v_n(\vec{r})) &= \langle \hat{W}_n \cdot \nabla \mathcal{Y}_n(\vec{r}), v_n(\vec{r}) \rangle + \langle S_t(\vec{r}) \mathcal{Y}_n(\vec{r}), v_n(\vec{r}) \rangle + X(\mathcal{Y}_n(\vec{r}), v_n(\vec{r})) \\ B(\mathcal{Y}_n(\vec{r}), v_n(\vec{r})) &= \left\langle \hat{\mathbf{A}}_{S_{s,n,n'}}(\vec{r}) \mathcal{Y}_{n'}(\vec{r}), v_n(\vec{r}) \right\rangle \\ L(v_n(\vec{r})) &= \langle F_n(\vec{r}), v_n(\vec{r}) \rangle \end{aligned}$$

In equation 3.4, $X(\mathcal{Y}_n(\vec{r}), v_n(\vec{r}))$ represents a volumetric stabilization term for which we studied three formulations:

$$\begin{aligned} X_{SUPG}(\mathcal{Y}_n(\vec{r}), v_n(\vec{r})) &= \langle R_n(\vec{r}), t \hat{W}_n \cdot \nabla v_n(\vec{r}) \rangle \\ X_{GLS}(\mathcal{Y}_n(\vec{r}), v_n(\vec{r})) &= \langle R_n(\vec{r}), t \hat{W}_n \cdot \nabla v_n(\vec{r}) + S_t(\vec{r}) v_n(\vec{r}) \rangle \\ X_{LS}(\mathcal{Y}_n(\vec{r}), v_n(\vec{r})) &= \langle R_n(\vec{r}), \hat{W}_n \cdot \nabla v_n(\vec{r}) + S_t(\vec{r}) v_n(\vec{r}) - v_n(\vec{r}) \rangle \end{aligned} \quad (3.5)$$

The most obvious difference between the LS, SUPG and GLS is the lack of a τ factor. This factor in SUPG and GLS is used to introduce an element-wise stabilization to improve the condition number and reduce the non-physical oscillations. Of course without such a factor, the LS is observed to be well applicable to void regions due to excess artificial diffusivity, but it fails to address the underlying condition number problems associated with widely varying cross section values in the domain. In this regard, we dropped the LS as a potential option and focused the remainder of the work on studying the GLS and SUPG methodologies.

Because of limited funding this year, the above formulation is only implemented in a test code that solves a one-speed fixed source problem. While this does limit the problems that can be studied, we can use this test code to investigate most of the issues that would result when applied to a full scale heterogeneous reactor problem by carefully understanding the issues arising in PROTEUS-SN. In the current report, we only summarize the work that was included in a recently published paper [17].

3.1 A Test Problem to Verify Spatial Mesh Convergence

The first problem studied using the test code has an assumed manufactured solution for each direction n in the set of directions N , given as

$$\mathcal{Y}_n(\vec{r}) = \frac{(N+1)}{2} \sin\left(\frac{\pi x}{L}\right) \sin\left(\frac{\pi y}{L}\right) \sin\left(\frac{\pi z}{L}\right) \quad (3.6)$$

In Equation 3.6, L is the spatial length of the square 2-D domain, which has all void boundary conditions. For simplicity, a total macroscopic cross-section of 0.1 cm^{-1} and scattering ratio of 0.5 was used. The purpose of this test problem is to display the spatial convergence of the methodology and ensure that no problems arise with the formulation with progressive mesh refinement. In addition to SUPG and GLS, we also coded up the even-parity (EP) methodology that PROTEUS-SN is based upon along with the standard discontinuous Galerkin methodology (DG) that forms the bulk of all S_N -based transport solvers for the past 30 years.

Figure 3.1 shows the solution accuracy convergence of linear quadrilateral FEM meshes with the various formulations. Since the analytical solution for the problem is known *a-priori*, we can compute the global solution accuracy in L_2 norm via integrals with high order quadrature. It is evident from the plot that the DG formulation yields the correct numerical order of convergence as expected from theory, $O(h^2)$ where h is the element size. The stabilized Galerkin methods namely SUPG and GLS have comparable convergence to the DG scheme and are as accurate with lesser degrees-of-freedom (DOF). On the other hand, the even-parity formulation clearly has consistently more error although it maintains a nearly quadratic rate of convergence with uniform refinement. The initial slow convergence of the even-parity method has also been observed with PROTEUS-SN and is due to the poor evaluation of the double derivative integral that linear basis functions provide.

Figure 3.2 shows the convergence of cubic quadrilateral FEM meshes where the expected theoretical convergence rate is $O(h^4)$. Once again, all the formulations tested yield the expected convergence order as the mesh is uniformly refined. From both Figures 3.1 and (3.2), we can conclude that the implementation of the SUPG and GLS formulations are

consistently accurate and note that the SUPG and GLS methods produce identical convergence plots for all cases. This is an artifact of the choice of total cross-section $\Sigma_t < 1$ which suppresses the impact of the additional term in the GLS scheme. In numerical experiments, it was observed that the GLS stabilization yields much less numerical dispersion when a larger total cross-section is used and is generally more accurate than the SUPG method.

Figure 3.1. Spatial Convergence of Linear Basis Functions

Figure 3.2. Spatial Convergence of Cubic Basis Functions

3.2 A Test Problem to Investigate Condition Number Issues

The second test problem has been designed specifically to analyze the effect of material property variations on the condition number of the formulation, since truly

heterogeneous problems will have large jumps in material properties. The domain for the test case is a checkerboard layout of cross section data shown in Figure 3.3. An isotropic source is imposed in the central square as indicated by the “S” and unless otherwise specified, all discretizations are performed using finite elements with linear basis functions. The purpose of this test is to measure the condition number of the bilinear operator, which can provide insights about the particular methodology: 1) can solve the operator to obtain an accurate solution and, 2) the solution procedure can be computed with a reasonable amount of computational effort.

As an example, for the even-parity methodology, the inclusion of a low-density region has caused the condition number of the overall system to observably increase and while not precluding a meaningful solution, the computational cost involved in getting that solution dramatically increases. For all methodologies, the weak form of the streaming-collision operator is explicitly formed and its highest and lowest singular values are measured based on the singular value decomposition (SVD) method with QR deflation [20].

Figure 3.3. Checkerboard Problem Domain with Alternating Low (blue) and High (red) Material Cross Section

In general, we can state that any methodology with a high condition number ($>10^8$) has limited practical use. For all of the test cases we consider here, we take the DG-FEM methodology as the reference with respect to both the solution and “best” condition number estimate. We introduce the parameter (μ) as the ratio of largest to smallest cross-section as a gauge for this problem as the condition number can be shown to be a function of μ . Figure (3.4) plots the computed condition numbers of all four methodologies when the total cross section values are 1.0 (red) and 0.1 (blue) while Figure 3.5 plots the results for total cross section values of 1.0 (red) and 0.001 (blue).

Figure 3.4. Condition Number Variations Depending on $\mu=10$ for All Methods (DG, EP, SUPG, and GLS)

Figure 3.5. Condition Number Variations Depending on $\mu=1000$ for All Methods (DG, EP, SUPG, and GLS)

Focusing first on the EP methodology, one can easily see that the condition number becomes considerably worse from Figure 3.4 to Figure 3.5 increasing by a factor of 100 at 100000 DOFs. This is consistent with the observed behavior of PROTEUS-SN where the required iterations to converge a given problem transition from a few thousand to a few million. The condition number for the EP method increases as $O(h^2\mu^2)$, whereas the condition number of the DG operator is observed to be $O(h\mu^{0.25})$. This clearly indicates that the cross section values in even-parity based codes (and by extension self-adjoint angular flux or dual

parity codes) are strongly restricted which translates to serious consequences when modeling heterogeneous geometry. The SUPG and GLS stabilized approaches perform nearly identical and have a comparable condition number with our reference DG operator. Similar to DG, we can estimate the condition number of SUPG and GLS to proportionally change as $O(h\mu^{0.25})$.

From Figure 3.4 and 3.5, it is clear that SUPG and GLS forms offer considerable improvements over the even-parity methodology and are a competitive alternate to DG formulation. This does not mean that SUPG and GLS can unconditionally give better solutions than DG, but only that the condition number issues we experienced with the even-parity based PROTEUS-SN can be resolved by switching the formulation to SUPG or GLS. There is still considerable research that needs to be performed on these methods to determine the optimal amount of stabilization needed in the context of routine reactor analysis problems where the DG and the EP methodologies have been validated before. Unfortunately, further research efforts will not be part of NEAMS scope due to the recent change in focus.

3.3 Assessment of the SUPG and GLS Operators

The even-parity (EP), self-adjoint angular flux (SAAF), stabilized upwind Petrov-Galerkin (SUPG), least squares (LS), and generalized least squares (GLS) methodologies can all be represented using a similar set of element-wise operators. First, we define a modified version of equation 3.3 to encompass all forms including SAAF (but not EP) as

$$r_n(\vec{r}) = \hat{W}_n \cdot \nabla y_n(\vec{r}) + S_t(\vec{r}) y_n(\vec{r}) - X_n(\vec{r}) - S_n(\vec{r}) = 0. \quad (3.7)$$

We next assume each methodology has a weak form discretization for each element and direction of the form

$$\underline{r}_{e,n} = \underline{A}_{e,n} \underline{y}_{e,n} - \underline{S}_{e,n} \gg 0, \quad (3.8)$$

where a single underline denotes a vector and a double underline denotes a matrix. In this case, we can generically define the common element-wise coefficient matrix as

$$\underline{A}_{e,n} = \left(S_{t,e} + t a_1 S_{t,e}^2 \right) \underline{F}_{K,L} + t \hat{a} \hat{W}_{n,K} \hat{W}_{n,L} \underline{P}_{K,L} + \left(a_4 + t a_2 S_{t,e} \right) \hat{a} \underline{W}_{n,K} \underline{U}_{e,K}^T \\ + t a_3 S_{t,e} \hat{a} \hat{W}_{n,K} \underline{U}_{e,K} + \hat{a} a_5 d_{g\vec{v}BC} \hat{W}_n \hat{n}_{g,e} \underline{W}_{e,g}, \quad (3.9)$$

where the constants needed for each method are defined in Table 3.1.

Table 3.1. Primary Coefficient Matrix Constant Definitions for Various Methodologies

Methodology	τ	a_1	a_2	a_3	a_4	a_5
SAAF (self-adjoint angular flux)	1	0	0	0	0	1
SUPG (Stabilized Upwind Pet. Gal)	τ	0	1	0	1	1
GLS (Galerkin Least Squares)	τ	1	1	1	1	1
LS (Least Squares)	1	1	1	1	0	$\Sigma_{t,e}$
SGS (Sub-Grid least Squares)	τ	1	1	-1	1	1

In its present form, PROTEUS-SN assembles the element matrices defined by equation 3.9 as part of its preconditioner and applies them as part of a matrix-free CG based solution algorithm. At issue is that the performance of the assembled matrix-vector preconditioner system is very poor; it achieves a mere 3-5% of the peak performance of the machine and it consumes a bulk of the available memory on the computer which limits the modeling ability of PROTEUS-SN by restricting the fidelity of the cross section data and geometry. The approach taken for the matrix-free coefficient application was observed to have even worse peak performance as the approach taken was not focused on performance, but only functionality.

Part of the previous year's work was to start looking at replacing the inefficient solution algorithm of PROTEUS-SN with a spatial multi-grid preconditioner system [18]. In that work, we narrowed down the focus on doing tetrahedral tessellation of the 3D mesh with multiple levels of structured grid preconditioners based upon our earlier observations. In each level of the system, we utilize the operator defined by equation 3.9, but because we choose to limit the operator application to tetrahedral and structured hexahedral meshes, we can take advantages of the matrix properties to improve the computational efficiency. Taking tetrahedral elements as an example, the spatial matrices defined by equation 3.9 can be broken into the component pieces:

$$\underline{\underline{F}}_e \rightarrow c_e \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \quad (3.10)$$

$$\underline{\underline{U}}_{e,K} \rightarrow u_1 \cdot \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \end{bmatrix} + u_2 \cdot \begin{bmatrix} -1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 \end{bmatrix} + u_3 \cdot \begin{bmatrix} -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \end{bmatrix} \quad (3.11)$$

$$\underline{\underline{U}}_{e,K}^T \rightarrow u_1 \cdot \begin{bmatrix} -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + u_2 \cdot \begin{bmatrix} -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} + u_3 \cdot \begin{bmatrix} -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (3.12)$$

$$\begin{aligned} \underline{\underline{P}}_{e,K,L} \rightarrow & p_1 \cdot \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + p_2 \cdot \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + p_3 \cdot \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \\ & + p_4 \cdot \begin{bmatrix} 1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + p_5 \cdot \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + p_6 \cdot \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \quad (3.13)$$

As can be seen, the tetrahedral spatial matrices can be reduced into a series of Boolean matrices followed by scalar which are very sparse. While multiplications and additions are equivalent in terms of cost, one should note that on a row wise basis, there are only a few unique Boolean operations and we can further combine the coefficients as an option. The goal of course is to minimize the work done to apply the operator and the memory footprint required to process it.

3.4 Performance Results of the Tetrahedral F matrix

As part of the work to study this aspect, we investigated the performance issues associated with implementing the coefficient matrix-vector application in several matrix-free ways. Using PAPI (Performance Application Programming Interface) to instrument the source code, we began with a simplified version of equation 3.9, which we took the A matrix to have only one component matrix (the F matrix, mass matrix or T_1 matrix). Thus, the application of the simplified A matrix to a vector can be written as

$$\underline{Y}_{e,a,g} = \sigma_{e,a,g} \underline{T} \cdot \underline{X}_{e,a,g} . \quad (3.14)$$

Note that energy and angle are treated identically in this case. There are several ways to implement equation 3.14, the most naïve of which is to form and store the spatial matrix:

```
DO E = 1, NumElements
  DO V = 1, 4
    Y(V,E) = 0.0d0
    DO VV = 1, 4
      Y(V,E) = Y(V,E) + C(E)*F(VV,V,E)*X(VV,E) ! 3 flops
    END DO
  END DO
END DO
```

Figure 3.6. Matrix Based “Apply F” for a Single Angle and Single Energy Group

The equivalent form for matrix-matrix products is clearly better for higher order finite element approaches, but since we are only interested in linear tetrahedrons which have a constant Jacobian, the matrix-free equivalent can be written as:

```
DO E = 1, NumElements
  ! 2 loads to resolve (1,E) and 1 load to get X(1,E) = 3 loads
  ! 2 loads to resolve (E) and 1 load to get C(E)
  ! We have four lines of code which constitute
  ! (3)*4 + (3)*4 + (3      3      3      3)*4 = 84 loads
  Y(1,E) = C(E)*(X(1,E)+X(1,E)+X(2,E)+X(3,E)+X(4,E)) ! 5 flops and 1 store
  Y(2,E) = C(E)*(X(2,E)+X(1,E)+X(2,E)+X(3,E)+X(4,E)) ! 5 flops and 1 store
  Y(3,E) = C(E)*(X(3,E)+X(1,E)+X(2,E)+X(3,E)+X(4,E)) ! 5 flops and 1 store
  Y(4,E) = C(E)*(X(4,E)+X(1,E)+X(2,E)+X(3,E)+X(4,E)) ! 5 flops and 1 store
END DO ! 1 Int E++, 1 Branch, 1 load/store for E and 1 load for NumElements
```

Figure 3.7. Matrix-free “Apply F” for a Single Angle and Single Energy Group

In the above example, the F matrix is known to be the constant four by four matrix for linear tetrahedrons:

$$\underline{\underline{F}}_e = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}. \quad (3.15)$$

The constant values are substituted in rather than stored for every element in the matrix-free form. However, to optimize vectorization, the ideal approach should reverse the index ordering to have the “Element” index on the inside:

```
DO E = 1, NumElements
  Y(E, 1) = C(E) * (X(E, 1) + X(E, 1) + X(E, 2) + X(E, 3) + X(E, 4)) ! 5 flops
  Y(E, 2) = C(E) * (X(E, 2) + X(E, 1) + X(E, 2) + X(E, 3) + X(E, 4)) ! 5 flops
  Y(E, 3) = C(E) * (X(E, 3) + X(E, 1) + X(E, 2) + X(E, 3) + X(E, 4)) ! 5 flops
  Y(E, 4) = C(E) * (X(E, 4) + X(E, 1) + X(E, 2) + X(E, 3) + X(E, 4)) ! 5 flops
END DO
```

Figure 3.8. Alternate Matrix-free “Apply F” for a Single Angle and Single Energy Group

Examining these three code implementations can be instructive as it gives us the basic understanding of how the compiler is modifying the code and how PAPI measures performance. In all three cases, the number of flops is easy to determine as the hardware counter only considers floating point operations. The loads and stores are much more complicated and we only can assert some aspects of what is required to get data. For any of the above implementations, we can state:

- Access of C(E) requires 1 base pointer address load, 1 offset value to load, and 1 load for the actual array value (total of 3 loads).
- Access of X(1,E) or Y(1,E) requires 1 base pointer address load, 1 offset value to load, and 1 load for actual array value (total of 3 loads). There is additional pointer math that occurs which should involve an intermediate load and store, but there are several options for each version.
- Loading the index variables E and NumElements (total of 2 loads)

With these guidelines, we included the best idea of how the loads are distributed for the matrix-free scheme in Figure 3.7 for an un-optimized compilation. As will be shown shortly, our crude breakdown is not entirely accurate or representative, but it is reasonably close.

Using the Intel 12.1 compiler, the three coding approaches were compiled using -O0 (un-optimized) and the flops, loads, and stores were measured using the following PAPI (version 5.1.1) events on Theo, an Intel Xeon X5760 chip. We note that some of these are averages as the measurements in PAPI did fluctuate slightly depending upon runtime issues (i.e. random noise).

PAPI_FP_INS: Floating point instructions executed
PAPI_LD_INS: Load instructions executed
PAPI_SR_INS: Store instructions executed

Table 3.2. PAPI Measurements of Matrix Based “Apply F” (-O0)

Number of Elements	0 (Overhead)	1	2	3	4	1000
FLOPS	0	32	64	97	129	32080
Loads	57	368	679	990	1301	311066
Stores	35	81	127	173	219	46044

Table 3.3. PAPI Measurements on Matrix-free “Apply F” (-O0)

Number of Elements	0 (Overhead)	1	2	3	4	1000
FLOPS	0	20	40	60	80	20120
Loads	57	140	223	306	389	83064
Stores	35	40	45	50	55	5042

Table 3.4. PAPI Measurements on Alternate Matrix-free “Apply F” (-O0)

Number of Elements	0 (Overhead)	1	2	3	4	1000
FLOPS	0	20	40	60	80	20072
Loads	57	189	321	453	585	132064
Stores	35	41	47	53	59	6042

From Tables 3.2 through 3.4, we can see there is a clear improvement in the number of FLOPS from the matrix based approach to the matrix-free approaches, but virtually no difference between the two matrix-free approaches again noting that we observed spurious noise in the PAPI_FP_INS measurement. More important than the reduced number of FLOPS was the observed factor of 4 reduction in the loads between matrix-based and matrix-free methods, and the factor of 2.4 reduction for the alternate matrix-free one. The reduction in the number of stores is even more significant, yielding a factor of 9 reduction from the matrix based approach to the matrix-free approach and a factor of 7.7 reduction for the alternate matrix-free approach. This can be explained as the elimination of the initialization line and the extra do loop that appears in Figure 3.6 relative to Figures 3.7 and 3.8. We can summarize the data in Tables 3.2 through 3.4 in Table 3.5. We note that all three options display a poor bytes/flop ratio given that we desire the ratio f to be near or less than 1.0 as will be shown later. This ratio represents the amount of data to be moved per floating point operation. In future architectures, the expense of memory movement will continue to outgrow the expense of floating point operations. We therefore wish to significantly reduce data loads and stores required per floating point operation.

Table 3.5. Summary of PAPI Measurements of -O0 for “Apply F”.

Version	Standard	Matrix-Free	Alternate Matrix-Free
FLOPS / element	32	20	20
Loads / element	311	83	132
Stores / element	46	5.0	6.0
(Loads+Stores)/FLOPS	11.2	4.4	6.9
$f = \text{bytes/flop}$	90	35	55

With respect to our estimated number of flops, loads, and stores in Figure 3.7, Table 3.5 indicates that the flops and stores are consistent with our understanding of the algorithm while the total number of loads (86 versus 83) is somewhat incorrect. This is understandable given that we did not look at the object or assembly level coding generated by the compiler and it is not clear how the pointer math is being done. The increased number of loads seen in Table 3.4 (alternate matrix-free) compared with Table 3.3 (matrix-free), can be attributed to the compiler being able to reuse the pointer math variables rather than calculating each pointer address uniquely.

The preceding measurements were done to ensure that we could understand where the loads, stores, and flops were coming from the actual source code we were writing. From here, we investigated the optimized results for the same three pieces of coding. Tables 3.6 through 3.8 provide the same PAPI measurements for the three pieces of coding using “-O3” and Table 3.9 summarizes the data similar to that done in Table 3.5.

Table 3.6. PAPI Measurements of Matrix Based “Apply F” (-O3)

Number of Elements	0 (Overhead)	1	2	3	4	1000
FLOPS	0	32	32	64	64	16005
Loads	56	92	156	191	251	48067
Stores	31	39	75	82	114	20042

Table 3.7. PAPI Measurements on Matrix-free “Apply F” (-O3)

Number of Elements	0 (Overhead)	1	2	3	4	1000
FLOPS	0	7	16	21	28	7060
Loads	53	61	71	80	89	9060
Stores	28	31	36	40	44	4035

Table 3.8. PAPI Measurements on Alternate Matrix-free “Apply F” (-O3)

Number of Elements	0 (Overhead)	1	2	3	4	1000
FLOPS	0	20	20	37	41	10335
Loads	58	62	64	73	69	3064
Stores	33	36	41	46	48	4039

Table 3.9. Summary of PAPI Measurements of -O3 for “Apply F”.

Version	Matrix Based	Matrix-Free	Alternate Matrix-Free
FLOPS / element	16	7.1	10
Loads / element	48	9.1	3.1
Stores / element	20	4.0	4.0
(Loads+Stores)/FLOPS	4.3	1.8	0.71
$f = \text{bytes/flop}$	34	14	5.7

To begin, the PAPI overhead between optimized and unoptimized compilations is nearly constant and thus can be ignored. From there, we can consider the matrix based results in Table 3.6 and compared against its companion results in Table 3.2. One can clearly see that optimization (vectorization) of the matrix-based version reduces the flops and stores by a factor of 2 from the un-optimized version, and the loads by a factor of 6.5. While this is good, at best it brings us to the performance of the un-optimized version of the matrix-free approach with respect to f . The matrix-free approach, which is the equivalent of hand unrolling the matrix based approach and hard wiring the matrix coefficients, shows comparable improvement in performance to the matrix based approach. The number of flops is reduced by 2.3, the loads by 9.2, and the stores are relatively unchanged. The most improved approach is the alternative matrix-free approach which sees a factor of 2 reduction in flops, a 42-fold reduction in loads and 1.5-fold reduction in stores. Comparing the numbers to each other in Table 8, we can clearly see the ratio f is approaching the desired 1.0 or less target but it is still considerably high.

To understand the importance of f , we must look at the performance which we define as

$$P = \frac{Flops}{Time(s)}, \quad (3.16)$$

where the best possible performance is the ratio of the clock speed $R_{cycle} \approx 2GHz$ to the clock

cycles per flop of $T_f \approx 1 \cdot \frac{cycles}{flop}$

$$P_{Peak} = \frac{R_{cycle}}{T_f} \approx 2 \frac{Gflops}{s}. \quad (3.17)$$

The number of flops is generally fixed given a targeted problem size and a given algorithm, but the time required to execute those flops is dependent upon how the problem size and algorithm coding interacts with the machine hardware.

One way of estimating the time required to execute a given number of flops is to use

$$Time(s) = Flops \cdot T_f + t_1 \cdot Loads + \sum_{j=1}^{J-1} (t_{j+1} - t_j) \cdot M_j, \quad (3.18)$$

which assumes $J-1$ levels of cache with “ J ” inferred as the main memory index of the machine. In equation 3.18, t_1 represents the average time required to retrieve a single floating point value from L_1 cache (the closest cache to the core processing unit) and load it into the CPU register. Similarly, the variables $t_2, t_3, \dots, t_{J-1}$ represent the average time required to retrieve a single floating point value from the higher level caches and transfer it to the CPU register. The value M_j stated simply is the total number of cache misses that occur in Cache j as a result of performing “Loads” worth of data from memory into the registers such that Flops worth of work can be done. The advantage of this performance model is that it allows one to abstract how an entire code actually interacts with the machine memory hierarchy. It therefore relies upon multiple measurements to determine values for $t_1, t_2, \dots, t_J$. While this is a great idea to use for a full code, our usage of them for a single subroutine is not as good because these numbers are fundamentally tied to the memory usage of each subroutine within

a broader code and the typical measurements are generic in their purpose. As such, this performance model does not directly inform us as to the importance of our specific coding decisions, but allows us to assess them in a generic fashion.

Another way to think about the performance model is to build it up from the ideal case to the most complicated level. First we break a given piece of coding into a series of floating point instructions I and assume the memory needed by those instructions is housed entirely in L_1 cache which is the case for the measurements done in Table 3.2 through Table 3.8. In this case we can write the time required to execute those instructions as:

$$Time(s) = \sum_{i=1}^I (L_i \cdot t_l + F_i \cdot T_f + S_i \cdot t_s). \quad (3.19)$$

In equation 3.19, L_i represents the number of loads per floating point instruction while F_i represents the number of floating point operations there are per instruction (single operation versus a vector operation) and S_i represents the number of stores per instruction. In this form, we assume the time required to perform a load and store are equivalent although we note that there are architectures where a store requires less time than a load. We also point out that for any given instruction there can be no flops, no loads, and no stores and equation 3.19 is just a generic representation. In this approach we can account for how the coding interacts with the memory hierarchy with regard to reuse of memory over multiple consecutive floating point instructions by including a unique counter of loads and stores for each instruction. As an example, we can consider a coding instruction from the top of Figure 4.3 to be broken into the three possible load/store/flop scenarios as shown in Figure 3.9.

```
Y(i) = C(i)*X(i) + B(i)*X(i) ! 12 loads, 1 store, & 3 flops
! or
temp = C(i)*X(i)           ! 7 loads, 1 store, & 1 flop
temp2= B(i)*X(i)           ! 4 loads, 1 store, & 1 flop (X is still in register)
Y(i) = temp + temp2        ! 4 loads, 1 store, & 1 flop (temp2 is still in register)
! or
temp = C(i)+B(i)           ! 5 loads, 1 store, & 1 add
Y(i) = temp * X(i)         ! 4 loads, 1 store, & 1 flop (temp is still in register)
```

Figure 3.9. Example Coding Instruction and Load/Store/Flop Implementations

As seen in Figure 3.9, we can deduce several floating point instructions and approaches to using memory based on assumptions about how the machine will translate our lines of code. From this example, we can also see how memory reuse in the register could impact the net number of loads and stores. What actually transpires from the coded instruction into the potential paths of floating point operations is of course platform and compiler dependent, and this example demonstrates that we cannot realistically build a performance model without understanding how the coded instruction is actually translated. The large differences between un-optimized results (Table 3.2 through Table 3.4) and optimized results (Table 3.6 through Table 3.8) indicate how difficult it is to accomplish this task.

In the previous section, we used PAPI to measure the actual loads, stores, and flops resulting from a sequence of coding statements on a given platform. We can translate this from a targeted set of instructions in a subroutine represented by equation 3.19 to an average relationship as

$$Time(s) = L_a \cdot t_1 + F_a \cdot T_f + S_a \cdot t_1. \quad (3.20)$$

We note how similar this is to the first two terms in the performance model of equation 3.18, but rather than use it in that way, we want to equate it to the measured behavior of our routine and thus define the intermediate constant

$$\frac{1}{P_a} = \frac{L_a + S_a}{F_a} \cdot t_1 + T_f \approx f \cdot t_1 + T_f \rightarrow P_a = \frac{1}{f \cdot t_1 + T_f}, \quad (3.21)$$

where P_a is the flops/cycles for the test subroutine when the memory fits entirely within L_1

cache. If we assume $t_1 \approx 1 \frac{\text{cycle}}{\text{byte}}$, then we can re-summarize the performance of our earlier

routine as shown in Table 3.10. Comparing this with the peak performance of 2 GFlops/s, we can see why the desired goal is to minimize f in equation 3.21 as our best routine gets a mere 15% of the peak performance ($0.3/2 = 0.15$)

Table 3.10. GFlops/s Estimate Using the Simplistic L_1 Cache Performance Model.

	Version	Matrix Based	Matrix-Free	Alternate Matrix-Free
<i>Un-optimized</i>	f	90	35	55
	GFlops/s	0.02	0.05	0.04
<i>Optimized</i>	f	34	14	5.7
	Gflops/s	0.05	0.13	0.30

The preceding analysis is itself a small amount of the work that goes into optimizing a code. We note the performance model is not complete and has not been directly compared against the measured flop rate on the machine for the subroutine. Because of the complexity of the proposed algorithm for rebuilding PROTEUS-SN and the time required to study just the simple preceding subroutine, one can understand the actual rebuilding process will take a considerable amount of time. In the end, while it was easy to get an initial version of PROTEUS-SN working by using the off-the-shelf product PETSc [19], to meet the functional needs of a NEAMS coupled dynamics product one has a considerable amount of research to carry out to build a quality production level tool that can compete with the plethora of existing reactor analysis tools available to potential users.

4 Conclusions

For coupled multiphysics time-dependent simulations, a quasi-static kinetics methodology was derived and added to the PROTEUS-SN code. In the method, the point kinetics parameters are updated at each time step via a steady-state forward and adjoint flux calculation. This kinetics methodology should be sufficient to meet the needs of current NEAMS transient analysis until a more rigorous kinetics methodology can be implemented. An initial verification test was performed using a two-dimensional numerical benchmark problem in which the control rod was inserted in 0.04 sec and the power was traced up to 100 sec. By comparing the kinetics parameters with the conventional methods, the transient capability of PROTEUS-SN has been verified to be working correctly.

In addition to building a transient capability, we spent time researching an alternate methodology for the PROTEUS-SN code based upon Stabilized Upwind Petrov-Galerkin (SUPG) and Generalized Least Squares (GLS) methods. The fundamental problem with the even-parity methodology is the increase in the condition number associated with mesh refinement and low density regions. A test code utilizing PETSc was built to compare the SUPG and GLS formulations against the even-parity and the standard discontinuous-Galerkin methodologies. The results of this work reproduced the observed problems with the even-parity methodology and demonstrated that both the SUPG and GLS schemes can avert the condition number issues seen in the even-parity method. Further, the SUPG and GLS schemes were shown to be consistent with the discontinuous Galerkin method in that neither has a distinct advantage with respect to accuracy or condition number although our test problems for accuracy estimations were simple. In summary, both the SUPG and GLS continuous finite element schemes pose valid research alternatives to the even-parity methodology embedded in PROTEUS-SN.

The final piece of work done was to demonstrate the effort required to take the PROTEUS-SN code from its current limitation of 10^{12} degrees of freedom with inefficient usage of the computing resources to the 10^{14} degrees of freedom with efficient usage of the computing resources. In our work, we focused on rewriting a conventional matrix-vector operation which has notoriously high memory requirements and poor computing efficiency with an alternative matrix-free scheme that improves the overall efficiency by a factor of 6. The conclusions we can draw from this work is that the optimization of PROTEUS-SN solver with the stabilized Galerkin methods and matrix-free approaches will need further research effort, but significant reductions in memory storage and improvements in performance can be achieved such that the large dynamics problems of interest can be accomplished in the near future.

References

1. M. A. Smith, A. Mohamed, A. Marin-Lafleche, E. E. Lewis, K. Derstine, C. H. Lee, Wollaber and W. S. Yang. FY2010 Status Report on Advanced Neutronics Modeling and Validation, ANL-GenIV-149 (September 2010).
2. M. A. Smith, D. Kaushik, A. Wollaber, W. S. Yang, B. Smith, C. Rabiti, G. Palmiotti, "Recent Research Progress on UNIC at Argonne National Laboratory," International Conference on Mathematics, Computational Methods & Reactor Physics, Saratoga Springs, New York, May 3-7 (2009).
3. M. A. Smith, A. Mohamed, A. Marin-Lafleche, E. E. Lewis, K. Derstine, C. H. Lee, A. Wollaber, and W. S. Yang, "FY2010 Status Report on Advanced Neutronics Modeling and Validation," ANL-GenIV-149, September (2010).
4. M. A. Smith, A. Mohamed, C. H. Lee, A. Marin-Lafleche, and W. S. Yang, "Zero Power Fast Reactor Deterministic Modeling with Heterogeneous Geometry," American Nuclear Society, Hollywood, FL, June 26-30 (2011).
5. A. Mohamed, W. S. Yang, M. A. Smith, and C. H. Lee, "Spectrum Interaction Effects on Calculating Foil Reaction Rates in ZPR Assemblies," International Conference on

- Advances in Mathematics, Computational Methods, and Reactor Physics (M&C), Rio de Janeiro, Brazil, May 8-12 (2011).
6. E. R. Wolters and M. A. Smith, "Scoping Analysis of the Advanced Test Reactor Using SN2ND," ANL-12-31, June (2012).
 7. A. F. Henry, "The application of Reactor Kinetics to the Analysis of Experiments," *Nucl. Sci. Eng.*, **3**, p 52-70 (1958).
 8. K. O. Ott and D. A. Meneley, "Accuracy of the Quasi-static Treatment of Spatial Reactor Kinetics," *Nucl. Sci. Eng.*, **36**, p 402-411 (1969).
 9. J. Devooght and E. Mund, "Generalized Quasi-static Method for Nuclear Reactor Space-Time Kinetics," *Nucl. Sci. Eng.*, **76**, p10-17 (1980).
 10. M. A. Smith, W. S. Yang, A. Mohamed, E. E. Lewis, "Perturbation and Sensitivity Tool Based on the VARIANT Option of DIF3D," American Nuclear Society Transactions **107**, San Diego, CA, Nov. 11-15 (2012).
 11. K. L. Derstine, "DIF3D: A Code to Solve One-, Two-, and Three-Dimensional Finite-Difference Diffusion Theory Problems," ANL-82-64, Argonne National Laboratory (1984).
 12. G. Palmiotti, E. E. Lewis, and C. B. Carrico, "VARIANT: VARIational Anisotropic Nodal Transport for Multidimensional Cartesian and Hexagonal Geometry Calculations," ANL-95/40, Argonne National Laboratory (1995).
 13. I. Dilber and E. E. Lewis, "Variational Nodal Methods for Neutron Transport," *Nucl. Sci. Eng.*, **91**, 132 (1985).
 14. C. B. Carrico, E. E. Lewis, and G. Palmiotti, "Three-Dimensional Variational Nodal Transport Methods for Cartesian, Triangular, and Hexagonal Criticality Calculations," *Nucl. Sci. Eng.*, **111**, 168 (1992).
 15. M. A. Smith, N. Tsoulfanidis, E. E. Lewis, G. Palmiotti and T. A. Taiwo, "Higher Order Angular Capabilities of the VARIANT Code," *Trans. Am. Nucl. Soc.*, **86**, 321-322 (2002).
 16. W. S. Yang, M. A. Smith, G. Palmiotti, and E. E. Lewis, "Interface Conditions for Spherical Harmonics Methods," *Nucl. Sci. Eng.*, **150**, 259 (2005).
 17. Vijay S. Mahadevan, Micheal A. Smith, "Stabilized Finite Element Schemes for the Consistent Discretization of the Neutron Transport Equation," American Nuclear Society Transactions **109**, Washington, D.C., Nov 10-14 (2013).
 18. Vijay S. Mahadevan, Michael A. Smith, "Scalable Multi-grid Preconditioning techniques for the even-parity SN solver in UNIC," International Conference on Advances in Mathematics, Computational Methods, and Reactor Physics (M&C), Rio de Janeiro, Brazil, May 8-12 (2011).
 19. S. Balay, K. R. Buschelman, W. D. Gropp, D. K. Kaushik, M. G. Knepley, L. C. McInnes, and B. F. Smith, "PETSc home page," www.mcs.anl.gov/petsc.
 20. J.G.F. Francis, "The QR Transformation, I", *The Computer Journal*, **4**, No. 3, p 265-271 (1961).

Nuclear Engineering Division

Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC