INL/EXT-21-64272
ANL/NSE-21/51

Performance Improvements to the Griffin Transport Solvers

September 2021

Yaqi Wang[1], Changho Lee[2], Yeon Sang Jung[2], Zachary Prince[1], Joshua Hanophy[1], Logan Harbour[1], Hansol Park[2], Javier Ortensi[1]

[1]*Nuclear Science and Technology Division, Idaho National Laboratory*
[2]*Nuclear Science and Engineering Division, Argonne National Laboratory*

Idaho National Laboratory

Argonne NATIONAL LABORATORY

INL/EXT-21-64272
ANL/NSE-21/51

Performance Improvements to the Griffin Transport Solvers

Yaqi Wang[1], Changho Lee[2], Yeon Sang Jung[2], Zachary Prince[1], Joshua Hanophy[1], Logan Harbour[1], Hansol Park[2], Javier Ortensi[1]

[1]Nuclear Science and Technology Division, Idaho National Laboratory
[2]Nuclear Science and Engineering Division, Argonne National Laboratory

September 2021

Argonne National Laboratory
Idaho National Laboratory

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under UChicago Argonne, LLC
Contract DE-AC02-06CH11357
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Page intentionally left blank

ABSTRACT

Griffin is a Multiphysics Object-Oriented Simulation Environment (MOOSE) based reactor multiphysics analysis application jointly developed by Idaho National Laboratory and Argonne National Laboratory. Griffin includes a variety of deterministic radiation transport solvers for fixed source, k-eigenvalue, adjoint, and subcritical multiplication, as well as transient solvers for point-kinetics, improved quasi-static, and spatial dynamics. A code assessment performed in FY-20 identified two significant issues with the transport solvers in Griffin: first, the primary heterogeneous SN (discrete ordinates) transport solver based on continuous finite element methods required significant mesh refinement and higher memory usage compared to solvers based on the method of characteristic for equivalent accuracy. Second, the homogeneous PN (spherical harmonics expansion) transport solver did not adequately support spatial polynomial refinement, which is a feature usually required for problems with spatial homogenization and pronounced streaming, typical in fast or gas-cooled reactor systems. To address the first issue, the development effort focused on the more promising discontinuous finite element method (DFEM)-based SN transport solver in Griffin. The addition of an asynchronous parallel transport sweeper and coarse mesh finite difference (CMFD) acceleration have rendered a superior heterogeneous SN transport capability for multiphysics problems that requires far less computing resources in terms of both CPU time and memory usage. This is demonstrated with typical thermal- and fast-spectrum reactor benchmark problems, including 2D Transient Reactor Test (TREAT), 3D Advanced Burner Test Reactor (ABTR), and 2D and 3D Empire microreactor. Last year, it was challenging to run the 3-D Empire eigenvalue problem with Griffin. Currently, Griffin can execute the problem in 20 minutes on 1,536 CPUs with roughly 80% parallel efficiency on the INL Sawtooth machine. For the second issue, the development effort focused on a new transport solver based on the hybrid finite element PN method (HFEM-PN), equivalent to the variational nodal method, as well as a new diffusion solver based on HFEM-Diffusion. This solver is intended for homogenized domains with multiphysics coupling (i.e., supports mesh displacement, seamless temperature feedback, etc.). Initial calculations with the HFEM-Diffusion implementation show very good parallel efficiency for the residual evaluations with the 2D ABTR benchmark. A future development effort will be centered on further improvements to the CMFD, HFEM-PN, and DFEM diffusion solvers to ensure Griffin meets performance and software quality assurance requirements for advanced reactor design and analysis.

ACKNOWLEDGEMENTS

This work was funded under the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program managed by the Department of Energy Office of Nuclear Energy. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

The authors are thankful to the following MOOSE/libMesh developers: Derek Gaston, Fande Kong, Alexander Lindsay, Roy Stogner, Guillaume Giudicelli, and Jason Miller. Their continuing support in constructive design discussions, supportive framework development, the Griffin git repository, testing, documentation, distribution, etc. proved to be indispensable. We are also thankful to our users, especially critics, that push us to examine all aspects of these codes.

Page intentionally left blank

CONTENTS

FIGURES

TABLES

ACRONYMS

CFEM	continuous finite element method
DFEM	discontinuous finite element method
DoF	degrees of freedom
HFEM	hybrid finite element method
MOOSE	multiphysics object-oriented simulation environment
NDA	nonlinear diffusion acceleration
NEAMS	nuclear energy advanced modeling and simulation
NRC	Nuclear Regulatory Commission
PJFNK	preconditioned Jacobian-free Newton-Krylov
PN	spherical harmonics expansion
RB	red-black
SN	discrete ordinates
SQA	software quality assurance

Page intentionally left blank

1. INTRODUCTION

Griffin is a reactor multiphysics analysis application jointly developed by Idaho National Laboratory (INL) and Argonne National Laboratory (ANL) using the multiphysics object-oriented simulation environment (MOOSE) framework [1, 2]. Griffin's primary mission is to support the analysis of various advanced reactor designs, outside the scope of traditional commercial light-water reactor (LWR) designs, as proposed in the Griffin software development plan [3]. Griffin utilizes the MOOSE modules for solid mechanics, thermal fluids, fuel performance, heat transfer, etc. for coupled multiphysics simulations. Griffin can be natively coupled to any MOOSE-based applications to compute additional reactor data based on the specific physics of the coupled applications. Griffin is intended to provide a flexible, extendable, and robust tool for the analysis of various advanced reactor designs for both steady-state and time-dependent simulations. Griffin capabilities are structured into four components:

- Radiation transport methods – various solvers supporting multi-scheme and multi-radiation transport
- Reactor analysis methods – reactor-specific analysis tasks, such as depletion, fuel management, etc.
- ISOXML module – the object-oriented toolkit for the management of multigroup cross sections and decay/transmutation data
- Cross section preparation – processing nuclear data and generating multigroup cross sections.

These capabilities are under active development and are currently supported by the DOE nuclear energy advanced modeling and simulation (NEAMS) program. Griffin development follows the software quality assurance (SQA) procedure outlined by the MOOSE SQA planning document, INL's PLN-4005 [4]. Griffin development began as a combination of INL's MAMMOTH [5] and Rattlesnake [6, 7] codes and ANL's MC^2-3 [8] code in early 2020. The capabilities of ANL's PROTEUS code [9] have been gradually migrated into Griffin. This integration was a joint decision by INL and ANL in order to avoid duplicative development efforts and to concentrate expertise to better address modeling and simulations challenges emerging from industry and the Nuclear Regulatory Commission (NRC).

In the last half of Fiscal Year (FY) 2020 [10], several existing PROTEUS models were converted into Griffin models. During the model conversion and testing, various issues were identified with the two transport solvers based on the continuous finite element method (CFEM) spatial discretization: the discrete ordinates (SN) method of the self-adjoint angular flux formulation (SAAF-CFEM-SN) with nonlinear diffusion

acceleration (NDA) [11] and the spherical harmonics expansion (PN) method (CFEM-PN) [12]. Two of the issues were significant. First, the heterogeneous legacy transport solver typically required more mesh refinement compared to PROTEUS-MOC [9] for an equivalent accuracy. This requirement results in significantly higher memory usage, making solving some relatively large models difficult with moderate computing resources. Second, the CFEM-based transport solver did not adequately support p-refinement, the refinement through increasing polynomial expansion orders on local mesh elements. This feature is usually required for problems with spatial homogenization and pronounced streaming, typical of fast or gas-cooled reactor systems. In addition, this legacy solver also suffered from poor linear convergence. The initial integration of a cross section application programming interface (CSAPI) [13] in Griffin led by ANL in FY 2020 also indicates that our existing transport solvers have performance issues for the fixed source calculations required by CSAPI.

The solutions proposed for the two issues were enhancing the discontinuous finite element method (DFEM) SN based transport solver (DFEM-SN) and adding a new transport solver based on the hybrid finite element method (HFEM) PN discretization (HFEM-PN) [14].

Historically, the DFEM-SN solver was originally added into Rattlesnake for multi-scheme transport calculations [15] and for the demonstration of a flexible diffusion acceleration technique referred to as NDA (nonlinear diffusion acceleration) [16]. Kernels, boundary conditions, etc. were added for supporting the multiphysics calculations required by preconditioned Jacobian-free Newton-Krylov (PJFNK) [17] solvers. The DFEM-SN solver was used to showcase pebble tracking transport calculations for pebble-bed reactor analyses [18]. A primitive sweeper for the DFEM-SN transport equations was implemented to accelerate the calculation and perform the transport update. The sweeper is analogous to using forward substitution to solve a lower triangular matrix but has some important differences. It is a matrix-free method, and the correct ordering to solve the SN transport equation unknowns is found by ordering the mesh cells in a manner determined by the streaming directions. The sweeper used common techniques such as pre-assembling and storing some of the local finite element matrices and can perform the transport update with good efficiency on one processor. But in parallel, the described implementation of the sweeper could only do Jacobi-style parallel sweeping (i.e., lagging the out-going angular fluxes from the neighboring processor subdomains) with domain decomposition. When used as a preconditioner for the PJFNK solver and the transport update for diffusion acceleration with NDA, a Jacobi-style sweeper can degrade the convergence of the linear solve and the effectiveness of diffusion acceleration with an increased number of processors compared with a true

parallel sweeper.

NDA was implemented for DFEM-SN. The implementation was moderately flexible in the sense that the low-order diffusion system could reside either on the same mesh as the transport system or on a coarser mesh that is embedded in the fine mesh and that the shape functions of the diffusion system can be in a higher order than the 0th order, as in CMFD (coarse mesh finite difference). The NDA implementation followed the design of SAAF-CFEM-SN NDA [7] where the low-order diffusion calculation was treated as the main calculation for potentially strongly coupled multiphysics calculations. However, in the DFEM-SN NDA, we often use a coarse mesh on the diffusion side, with which we cannot couple other physics. These physics are often defined on the fine mesh, thus sophisticated data transfers are required. The diffusion system, by design, must have the same domain decomposition with the coarse mesh as the decomposition with the fine mesh for the transport system. This design often becomes a limiting factor on how many processors we can use because the number of coarse mesh elements can be smaller than the desired number of processors. Also, the domain partitioning based on the coarse mesh can cause a severe imbalance in parallel computing because the numbers of fine elements contained in all the coarse elements may vary drastically. On the user interface side, DFEM-SN NDA requires two input files each for the low-order diffusion system and the transport system. The two input files have some duplicated information that can confuse users. It is thus imperative to have a true parallel sweeper and to re-implement NDA, or sometimes referred to as CMFD when coarse mesh is employed, in a multiphysics environment. We expect that, with these improvements, the DFEM-SN solver can meet the needs for converting heterogeneous models in PROTEUS-MOC.

SAAF-CFEM-PN solvers were added in Griffin for multi-scheme transport calculations in the past. We observed similar drawbacks with SAAF-CFEM-SN solvers, such as often needing finer meshes. Slow convergence is often observed in certain calculations due to the lack of dedicated acceleration techniques. Although it is possible for us to improve SAAF-CFEM-PN to have a faster convergence, p-refinement for CFEM based methods is more fundamental. Thus, as consequence from these lessons learned, we decided to add a new solver based on HFEM-PN in Griffin. HFEM-PN, also known as VNM (variational nodal method) [14], has been implemented in the past in several codes, including DIF3D-VARIANT [19], PROTEUS-NODAL [20], INSTANT [21], etc. Both ANL and INL have the expertise for this transport solver. It has proved superior for solving problems with spatial homogenization and with pronounced streaming, typical of fast or gas-cooled reactor systems, because it allows for p-refinement and has a special fast iterative solving technique, called red-black (RB) iteration [22]. The new implementation in Griffin is

not simply code rewriting because we are aiming to also make this new solver support multiphysics calculations. The solver will allow the direct coupling of a three-dimensional mesh displacement and other feedback mechanisms due to temperature and density changes. The solver will have the same uniform user interface as all the other transport solvers in Griffin. The new implementation will follow our SQA procedure to ensure its long-term sustainability with the other capabilities in Griffin.

Griffin is a complex code system in the sense that it leverages a number of software packages shipped through the MOOSE framework. Besides computing hardware, operating systems, and compilers, it depends on MPI implementations, PETSc [23], SLEPc [24], libMesh [25] and various packages linked through them, such as METIS [26], ParMETIS [27], SuperLU_dist [28], MUMPS [29], HYPRE [30], which are all externally developed. Thus, it is critical to set up some integrated benchmark tests for Griffin to ensure a consistently good performance in terms of both memory usage and CPU time with the continuous developments of hardware/operating systems/compilers, the external packages, and MOOSE/Griffin. We started adding some performance benchmarks for the Griffin transport solvers while addressing the two aforementioned performance issues. These performance benchmarks can provide a figure of merit for measuring our code development efforts on top of code functionalities in the future. We note that these performance benchmarks cannot be replaced by regression tests in that the regression tests are mainly used for testing the correctness of Griffin capabilities.

In Section 2.1, we detail the implementation of the new parallel sweeper and the CMFD acceleration. Some related improvements, such as adding the multiplicative prolongation and the transport update with scattering are also discussed. In Section 2.2, we present the MOOSE framework improvements for supporting HFEM-PN. We will also talk about the kernels and the red-black iteration for HFEM-PN. In Section 3, we examine three performance benchmarks: Empire microreactor and Transient Reactor Test (TREAT) for DFEM-SN, and Advanced Burner Test Reactor (ABTR) duct-heterogeneous model for DFEM-SN and homongenous model for HFEM-PN. Significant performance improvements are shown to demonstrate the progress made during the joint-team development of Griffin for addressing the analysis needs of advanced reactor designs. In Section 4, we summarize our work and discuss potential further improvements on transport solvers.

2. GRIFFIN IMPROVEMENTS

2.1 Improvements for DFEM-SN

We improved the DFEM-SN solver in Griffin to meet the requirements for heterogeneous transport calculations. The improvements include: the implementation of an asynchronous parallel sweeper, the refactoring of CMFD, and new capabilities to make the diffusion acceleration more effective. These three improvements are detailed in the following three subsections.

2.1.1 The Parallel Sweeper

The transport operator denoted with $\mathbf{L}$ in the multigroup neutron transport equations is defined as the streaming operator plus the collision operator:

$$\mathbf{L} = \text{diag}\{\vec{\Omega}\cdot\vec{\nabla} + \Sigma_{\text{t},g}, \quad g = 1,\cdots,G\}, \tag{1}$$

where $\vec{\Omega}$ is the independent variable of the streaming direction, $\Sigma_{\text{t},g}$ is the total interaction cross section, and G is the total number of energy groups in the multigroup equations. *diag* means that the angular fluxes for all groups are decoupled in the transport operator. For more detail about the multigroup neutron transport equations, we refer readers to Ref. [7]. In the discrete ordinates method (SN), the transport operator is decoupled among all the streaming directions from a specific angular quadrature $\{\vec{\Omega}_m, m = 1,\cdots,M\}$,

$$\mathbf{L} = \text{diag}\{\vec{\Omega}_m\cdot\vec{\nabla} + \Sigma_{\text{t},g}, \quad m = 1,\cdots,M; g = 1,\cdots,G\}, \tag{2}$$

where M is the number of directions in the quadrature. A transport update is referred to as

$$\psi_{m,g} = \mathbf{L}_{m,g}^{-1} s_{m,g}, \quad m = 1,\cdots,M; g = 1,\cdots,G, \tag{3}$$

where $s_{m,g}$ is the angular source in direction m and of group g, $\psi_{m,g}$ is the angular solution in direction m and group g, and $\mathbf{L}_{m,g} \equiv \vec{\Omega}_m\cdot\vec{\nabla} + \Sigma_{\text{t},g}$. This transport update is used in source iterations for iteratively solving problems with scattering in which the angular fluxes in each direction and energy group are coupled. It is also used in PJFNK as a preconditioning step and in NDA for updating the angular fluxes and the closures for the diffusion system at every diffusion iteration. The transport update is a key component for solving the

multigroup neutron SN transport equations.

We can typically find an ordering of elements for each streaming direction $\vec{\Omega}_m$ in the DFEM with an upwinding scheme [31–33] on an unstructured mesh. This ordering is based on the upwinding dependency between neighboring elements and is dependent on the $\vec{\Omega}_m$ streaming direction. Essentially, the discretized transport operator or the transport matrix can be transformed into a block (element-wise) lower-triangular matrix based on the element ordering. Then we can invert the transport operator on local elements with the local source and the latest solutions of previously solved elements (upwind elements) and eventually obtain $\psi_{m,g}, g = 1, \cdots, G$ by sweeping through all the elements with the ordering. The sweep solver is matrix-free because we do not need to form the entire matrix of the transport operator for the transport update. There could be cases where we cannot find an ordering to make the transport matrix an exactly lower triangular matrix, such as with re-entering element faces, cyclic dependencies among elements, reflecting boundaries, etc. In these cases, we can break some dependencies from an element with its neighboring element to get an approximated inversion of the transport operator

$$\psi_{m,g} = \tilde{\mathbf{L}}_{m,g}^{-1} s_{m,g}, \quad m = 1, \cdots, M; g = 1, \cdots, G, \tag{4}$$

by lagging some angular fluxes, which are often referred to as significant angular fluxes, during an iteration within iteration schemes. The significant angular fluxes are known before starting a sweep and will be updated by the sweep. The approximated transport update is typically sufficient for the purposes of preconditioning or diffusion acceleration. The functionality that takes a source s and finds a solution ψ by performing the mesh sweeping is referred to as the sweeper. The sweeper is essential for solving the multigroup transport equations discretized with DFEM-SN.

Sweeping is analogous to solving a lower triangular matrix with forward substitution. Forward substitution is not generally a parallelizable algorithm. As discussed in the introduction, the previous sweeper in Griffin achieved parallel scalability by lagging flux information between processors and thus instead of forward substitution on a lower triangular matrix, the sweeper was performing an operation analogous to a Jacobi iteration. Although this leads to very good parallel efficiency, the convergence rate can become arbitrarily slow as more and more processors are used. The linear system resulting from the DFEM discretization of the SN equation is sparse. The sparsity allows sweeping to be parallelized without resorting to lagging flux information at the processor boundaries. One of the earliest works that described the potential

for sweeps to be massively parallel showed that the manner in which the domain is decomposed among the processors is a key factor in determining the parallel efficiency [34]. The potential for sweep solvers to scale to tens and hundreds of thousands of processors was further investigated for problems with uniform meshes in several more recent works [35, 36]. Critically, these works investigated problems where the domain decomposition was controlled to optimize sweeper performance. The domain must be decomposed into simple cube-like shapes. In Ref. [37], reactor assembly problems were investigated with unstructured meshes, but importantly, a semi-regular uniform-like domain partitioning was used and good scalability of the sweeps was shown to over one million processes. This recent work again clearly demonstrates that having a uniform partitioning of the domain facilitates sweeping unstructured meshes with good parallel efficiency.

Requiring that the domain partitioning be limited to simple cube shapes would be a severe limitation for Griffin. The sweep solver in Griffin does not control the decomposition of the domain. Therefore, the sweeper must work for domains of arbitrary shape. Developing a generally efficient sweep solver for such problems is still an area of active research. Recently, good parallel scalability to hundreds of thousands of processors was demonstrated for problems with unstructured domain partitioning [38], but this was achieved by lagging some flux information at processor boundaries. This type of sweep would generally exhibit better convergence behavior than the simple Jacobi style sweeper previously used in Griffin, but there are potential benefits to having a sweeper that does not need to lag any flux information. It is quickly mentioned here that we are not discussing the situation where there are cyclic dependencies in the mesh, but where the partitioning of the mesh requires either that flux information be lagged, or that multiple messages be passed back and forth between neighboring processors. The first work published that sought to sweep a generally unstructured mesh with unstructured partitioning [39] described an algorithm that was potentially massively parallel, but scaling results were only investigated in the work to 256 processors. A follow-up work expanded on the algorithm and included scaling results to several thousand processors [40]. The mesh investigated in this work had some uniform structures, but an unstructured partitioning of the mesh was investigated. The parallel efficiency reported to several thousand processors was better than 70% for some of the specific algorithms in the paper, giving evidence that sweeps can be efficient solvers for problems with unstructured partitions.

The sweeper now implemented in Griffin shares basic algorithmic similarities with the sweeper proposed in Ref. [39]. To quickly summarize, the basic algorithm involves forming a directed graph representing the angular flux dependencies among the cells and then using a breadth-first search to sort the graph, resulting

in the proper sweeper ordering. As discussed in Ref. [39], the problem encountered when domain decomposition is arbitrary is that many messages may need to be passed back and forth between processors. For the efficient communication of flux information between processors, the new sweeper in Griffin uses communication routines developed as part of the ray tracing module in MOOSE [41]. The communication routine attempts to group individual messages together to limit the number of messages that must be sent. The trade off is that, while collecting more individual messages into a larger message generally decreases the communication time, it increases the time that a neighbor processor might spend waiting for a message before it can start doing work. In this report, we call the new sweeper in Griffin the "asynchronous" sweeper. This name emphasizes that the sweeper requires neither a uniform mesh, nor a uniform mesh decomposition, nor any information lagging at processor boundaries. This "asynchronous" sweeper does not attempt to communicate flux information between processors in a single message. Instead, a processor may receive a single message or multiple messages and will start work whenever it can. The specific message a processor receives first is unknown ahead of time and may vary from run to run. Thus the message passing between processors is less synchronized than with some other algorithms. The new asynchronous sweeper currently implements a simple sweep scheduler based on the "first in first out" paradigm. Each processor maintains a list of directions for which it has received sufficient boundary information to begin work on. The directions are worked on preferentially in the order in which they become available. Since sweep fronts may become temporarily hung up at processor boundaries while processors hand messages back and forth, some simple logic is included to try and advance other directions when the preferential direction is delayed at a processor boundary.

Important aspects about the scalability of sweep solvers are discussed in Ref. [35], which includes the weak scaling law for growth with a solve time of $\mathcal{O}(dP^{1/d}+M)$, where P is the number of processors, M the number of directions in the angular quadrature set, and d the spatial dimension of the problem. Thus the sweepers only have polynomial scaling in time, which is not as good as some other common linear solver algorithms. But notice when M is large, that is when more angles are used, the scalablity of the sweeps won't be affected by the poor scaling in P until P becomes larger. The scaling law also says nothing about the efficiency of sweeps. Although other algorithms scale better than sweeps, a sweep solver usually remains a better choice on any reasonable number of processors, because sweeps have a very low computation cost.

Figures 1 and 2 show a weak scaling test of the new asynchronous sweeper on a uniform cubic mesh partitioned into uniform cubic domains. The first shows the "grind time" or solution time for the sweeper

divided by the number of unknowns. This plot is included in the report to provide a reference for the speed of the current asynchronous sweeper. These calculations were performed on the Sawtooth cluster at INL. Figure 2 shows the same information as the grind time plot, but reports the parallel efficiency directly. Figure 1 shows that the current sweeper is efficient, taking on the order of 100s of nanoseconds per unknown, but this number may be improved with further optimizations of the code implementation in the future. Two important features of these plots are that the sweeper becomes more efficient as more energy groups are solved and also as more directions are solved. The improvement in efficiency as more energy groups are solved is caused by the sweeper solving all energy groups at the same time in a given cell for a specific direction. The trends seen in the solve time versus the number of directions indicate that our current sweep scheduler exhibits the correct behavior, but these trends may also be improved in the future with more advanced sweep scheduling algorithms.

Although the uniform cubic mesh problems are useful for benchmarking purposes, a primary requirement of the multiphysics use case is that the sweeper work well on unstructured meshes and general mesh partitioning. We will later show strong scaling results from the sweeper for an unstructured mesh problem, and those results indicate reasonable performance. A second set of results are shown here based on a modified uniform problem. A uniform mesh of prisms is used as shown in Figure 4, but the partitioning is setup such that there is a jagged "sawtooth" shaped boundary between processors. This mesh will require many messages to be passed back and forth between processors to advance the sweep front. Weak scaling results for this mesh are shown in Figure 3. The grind time is still comparable to the uniform mesh results, thus pointing at the versatility of the new sweep solver.

In summary, this "asynchronous" sweeper is seamlessly integrated in Griffin for multiphysics simulations. It delivers reasonable scalability with arbitrarily decomposed unstructured meshes for multiphysics reactor analysis. It utilizes the parallel communication infrastructure developed for ray-tracing and a simple sweep scheduler. Both grind time and parallel scalability can be improved with further refinement of both the sweeping algorithms and the code implementation.

2.1.2 The CMFD Acceleration

We implemented a flexible NDA method in Griffin in the past [16]. This flexible NDA discretizes both the SN transport equation and the diffusion equation using DFEM. The method is flexible in that "the diffusion equation can be discretized on either the same mesh as for the transport equation or a coarser mesh

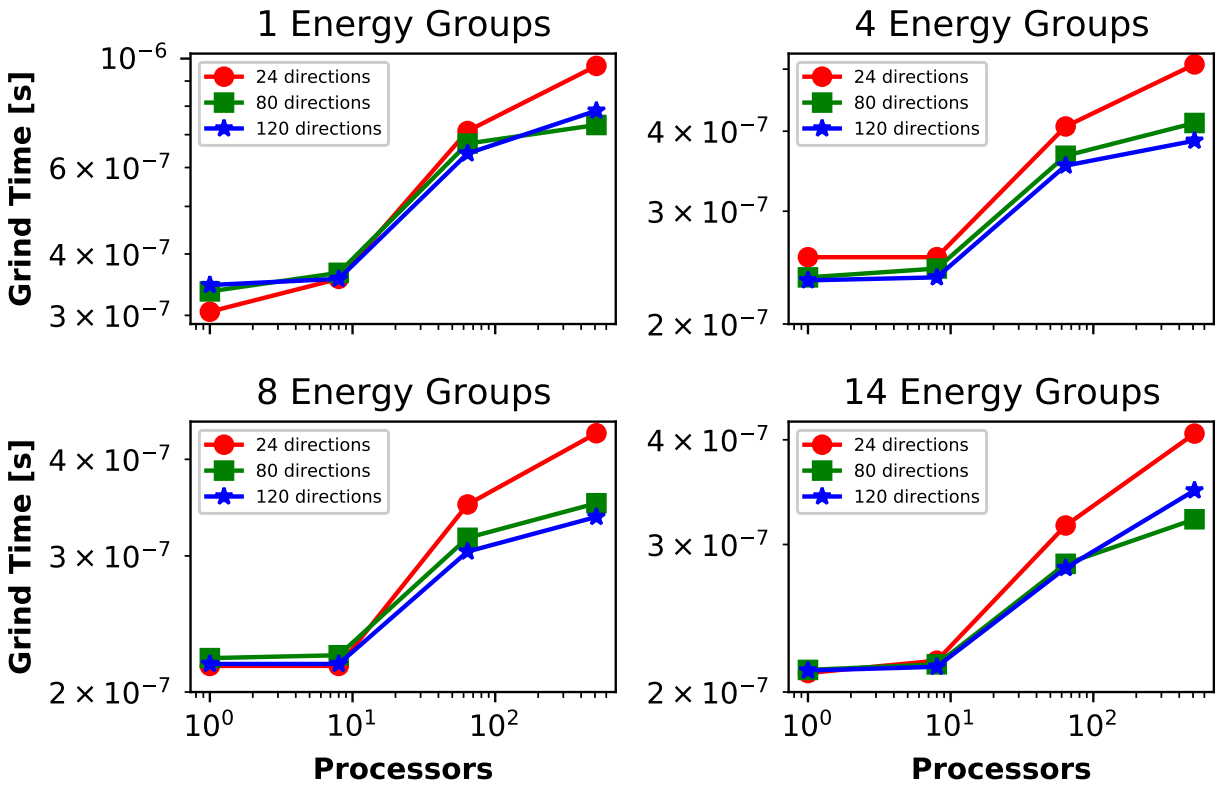

Figure 1: Plots showing the "grind time" or solution time for the sweeper divided by the number of unknowns. This is weak scaling to 512 processors for a uniform cube with uniform volumetric partitioning and 512 cells per processor.

with the only restriction that the coarse mesh is nested within the transport mesh" [16]. The FEM shape function orders of the two equations can be different as well. The consistency of the transport and diffusion solutions at convergence is ensured by using a projection operator mapping the transport into the diffusion FEM space. The implementation is similar to the NDA for SAAF-CFEM-SN in Griffin [11]. The low-order diffusion system is considered as the main application with which the problem is treated as a transport-corrected diffusion system. The diffusion acceleration is done through the Picard iteration between the low-order diffusion solve and the transport update while the diffusion equation is allowed to be strongly coupled with other physics. This is typically fine when the mesh used by diffusion system is identical with the mesh for transport system, referred to as fine-mesh NDA, but can cause trouble when a coarse mesh is employed because:

- The coarse mesh diffusion system cannot couple other physics directly in multiphysics calculations which require the fine mesh

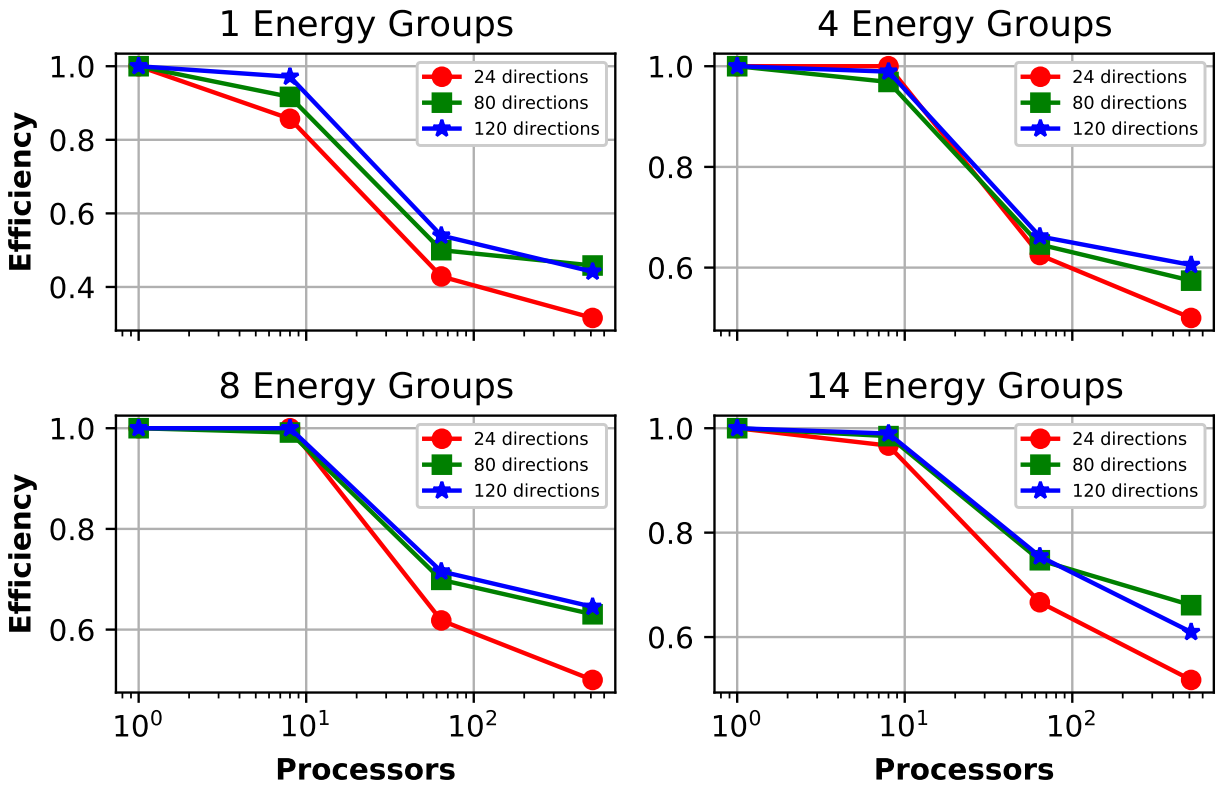

Figure 2: Plots showing the efficiency of the sweep solve. This result is for the same test as in Figure 1.

- The implementation requires the coarse and fine mesh to have the same domain decomposition thus the number of coarse elements can limit the number of processors we can use and a severe imbalance can be introduced because the numbers of fine elements contained in coarse elements may be quite different
- The implementation requires two input files for the diffusion system and the transport system, which share some redundant information that can confuse users
- It is difficult to integrate NDA with transient or depletion calculations with this design
- The coarse mesh must contain element types supported by libMesh. For example, 2D hexagon element is not allowed because libMesh currently does not support this element type.

Given all of these limitations on the initial implementation, we decided to re-implement the diffusion acceleration for DFEM-SN when coarse meshes are employed. In this re-implementation, we focus on a finite difference approach for the diffusion equation (i.e., CMFD), although a future extension with flexible

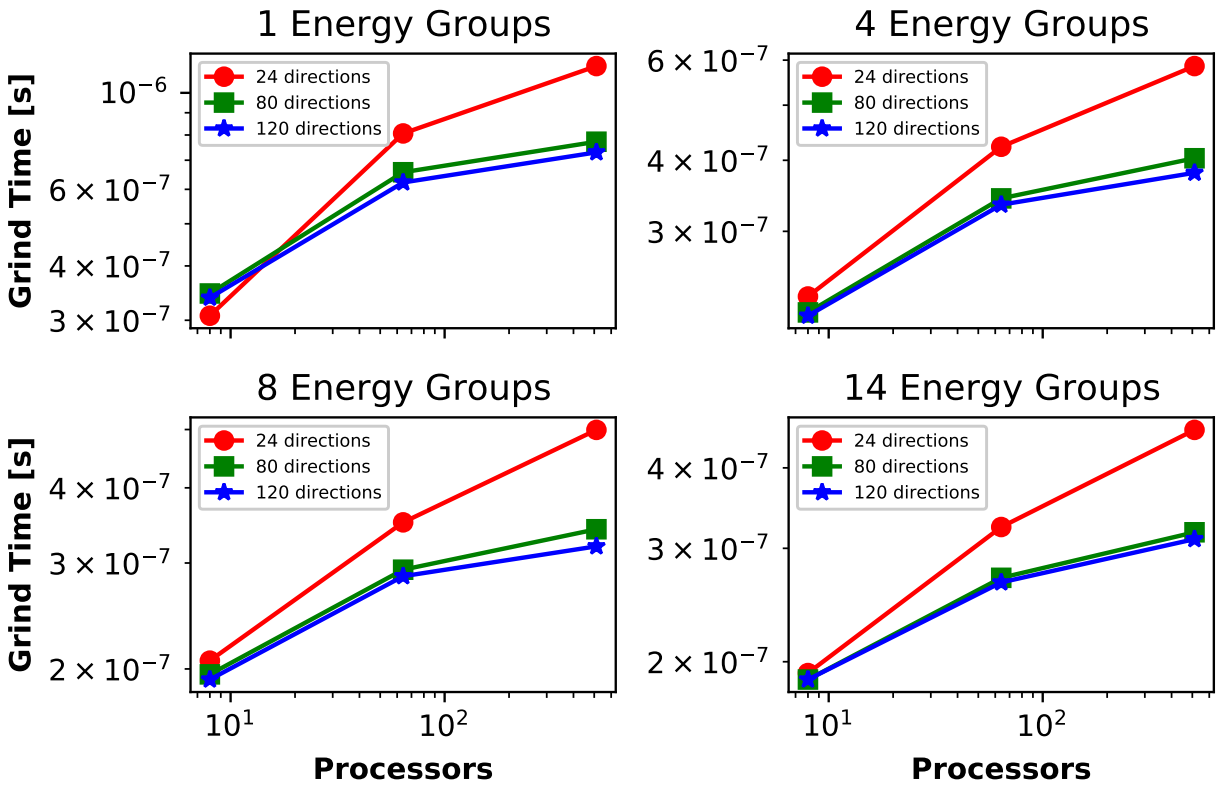

Figure 3: Plots showing the grind time of the sweep solve for the mesh with partitioning shown in Figure 4. This is weak scaling to 512 processors with 900 cells per processor.

NDA is possible. The major difference in this new implementation is that the diffusion system solver is treated as an acceleration step for solving the transport equation with Richardson (i.e., source) iterations. The projection, solve, and prolongation of CMFD are wrapped in a CMFD acceleration object that can be optionally turned on through the input. Multiphysics iterations can be turned on with this CMFD acceleration object. The implementation can take the advantage of the finite difference (DG0) approach and make the projection/prolongation more efficient. The implementation assembles the matrices for the finite difference system and directly calls SLEPc/PETSc for eigenvalue/nonlinear solves. The CMFD solve is parallelized by using PETSc parallel matrices and vectors. Griffin can automatically determine the number of processors to use for solving the CMFD system by limiting the number of degrees of freedom (DoF) per processor. The rule of thumb is to have each processor have more than 2,000 DoFs. Users can provide a parameter value to override the number of processors that would be automatically selected by Griffin. The implementation makes use of coarse element IDs assigned to all fine elements. The coarse element ID assignment can be quite flexible, in the sense that the coarse element can be in any shape and can neighbor an arbitrary

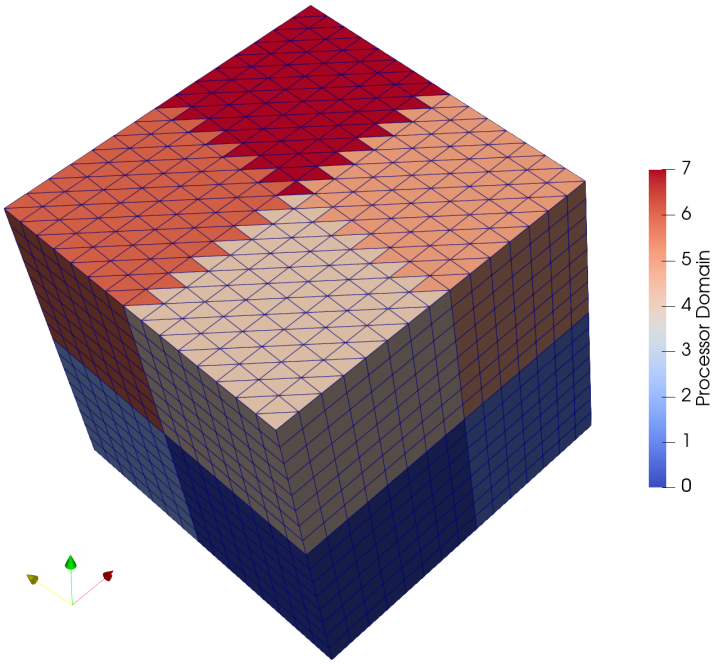

Figure 4: Uniform mesh partitioned such that the boundary between domains has a "sawtooth" shape.

number of coarse elements. The code even allows the interface between two neighboring coarse elements to be zigzagged. Currently this CMFD acceleration object does not support source problems or eigenvalue problems with the DSA scheme [42].

2.1.3 Other Improvements for DFEM-SN

When we obtain scalar fluxes on a coarse mesh with CMFD, we currently perform the prolongation with

$$\Psi_{m,g}(\vec{x}) = \Psi_{m,g}(\vec{x}) + \frac{\phi_g - \bar{\Phi}_g}{\sum_{m=1}^{M} w_m}, m = 1, \cdots, M; g = 1, \cdots, G, \tag{5}$$

for all fine elements, where ϕ_g is the CMFD solution on the coarse element, including the fine element, $\bar{\Phi}_g$ is the averaged fine scalar flux over all fine elements included in the coarse element, and w_m is the weight of the streaming direction $\vec{\Omega}_m$ of the angular quadrature. This prolongation formulation is labeled as the additive prolongation. We can see that, with this formulation, we only accelerate the 0-th angular flux moments (i.e., the scalar fluxes). The correction for the high-order angular flux moments is zero. We observed that another

type of prolongation, referred to as multiplicative prolongation:

$$\Psi_{m,g}(\vec{x}) = \Psi_{m,g}(\vec{x})\frac{\phi_g}{\bar{\Phi}_g}, m = 1,\cdots,M; g = 1,\cdots,G, \tag{6}$$

shows advantages for certain problems with significant anisotropic scattering. Although both formulations preserve the averaged scalar fluxes over coarse elements, the multiplicative prolongation also scales the high-order angular flux moments with the same scaling factor applied to the scalar fluxes. We added an option for users to perform the multiplicative prolongation in Griffin. This option is available for both fine-mesh NDA and the new CMFD implementation.

When the mesh used for the diffusion system is identical with the mesh for the transport system with fine-mesh NDA, Eq. (3) for the transport update can deliver an effective convergence for the Richardson iteration accelerated with the diffusion solve, where the source is constructed as the residual of the neutron transport equation

$$s = \mathbf{L}\Psi - \mathbf{S}\Psi - \frac{1}{k}\mathbf{F}\Phi, \tag{7}$$

where $\mathbf{L}$ is the full transport operator, $\mathbf{S}$ is the scattering operator, $\mathbf{F}$ is the fission operator, $\Phi = \int_{\mathscr{S}} \Psi d\,\Omega$ is the scalar flux, and k is the k-effective. The result of the transport update is then the correction of the angular flux, that is

$$\Psi \leftarrow \Psi - \bar{\mathbf{L}}^{-1}s. \tag{8}$$

However, a significant degradation of the convergence can be observed when coarse mesh for the diffusion system is employed. In such cases, we found that introducing the scattering operator in the transport update can be beneficial. The extreme case would be

$$\psi \leftarrow \psi - (\bar{\mathbf{L}} - \mathbf{S})^{-1}s. \tag{9}$$

Nevertheless, due to the cross-group scattering and the coupling for all directions with the in-group scattering, the reduced number of Richardson iterations can be overshadowed by the cost for performing this full inversion of transport plus scattering operator. Instead, we can perform the group sweeping for the down-

scattering, a limited number of in-group iterations for the in-group scattering, and thermal iterations for the up-scattering. We could also transform the system equation with angular flux moments and use GMRes to find an approximate inversion of the transport plus scattering operator. This capability of partially inverting the transport plus scattering operator was added to Griffin.

2.2 Implementation of the HFEM Solver

A new solver option based on the variational nodal method (VNM) [14] has been implemented in Griffin to support routine core design calculations with spatial homogenization and with pronounced streaming, typical of fast or gas-cooled reactor systems. In the finite element community, the VNM formulation is referred to as the hybrid finite element method (HFEM) [43] with PN (spherical harmonics expansion). Hereafter, VNM is referred to as HFEM-PN or HFEM for brevity and consistency. The HFEM weak formulation requires the introduction of additional variables acting on element surfaces compared to CFEM/DFEM formulation. These variables, commonly referred to as the *Lagrange multipliers*, enforce strict local neutron balance for individual elements while continuity across element interfaces is enforced weakly. The HFEM solution is composed of two parts: volumetric and surface solutions defined on each element and surface, respectively. For example, in the diffusion equation, the volumetric and surface components correspond to the scalar flux and the net current, respectively. One important characteristic of HFEM is that, unlike CFEM, the volumetric solution is not required to be continuous across element interfaces. This non-conforming property allows us to effectively solve the even-parity P_N transport equation on the homogenized coarse mesh structures with higher order polynomial representations of solutions.

In order to implement the HFEM solver, the base MOOSE framework was updated to incorporate degrees of freedom defined on element interfaces in the weak forms. We initially implemented the HFEM-diffusion solver in Griffin as a prototype to test the feasibility of the HFEM solver within the framework. There are two major capabilities required for the completion of the HFEM scheme: the even-parity P_N solver and the RB iteration scheme. In this fiscal year, we focused on the implementation of the even-parity P_N equation solver. The more efficient RB iteration scheme will be incorporated and completed in the next fiscal year.

The detailed implementation included: the improvements in the MOOSE framework for residual/Jacobian evaluations with Lagrange multiplier terms defined on element interfaces, adding kernels, boundary conditions, etc. and extending the action system to make HFEM-PN/Diffusion available to users, and adding

the RB iteration to solve the HFEM-PN/Diffusion equations more efficiently. The implementation is detailed in the following three subsections.

2.2.1 Improvements in MOOSE to Support HFEM-PN

In order to support the HFEM formulation in Griffin, the explicit incorporation of degrees of freedom defined on the element interface is required for using Lagrange multipliers to enforce the weak continuity across element interfaces. However, libMesh [25] currently does not support variables defined on element sides. To bypass this difficulty, we added the so-called lower-dimensional blocks for internal element sides and boundaries. With these lower-dimensional mesh blocks, the Lagrange multipliers can be defined as normal solution variables. We then extended the DG (discontinuous Galerkin) kernels that can bring in not only the variables defined on element and its neighbors across a side but also the quantities defined on the side in the lower-dimensional block.

The MOOSE system was updated to explicitly include the surface variables as degrees of freedom. The extended DG kernels can assemble the residuals corresponding to element, neighbor, and side together. They can also assemble the nine pieces of a Jacobian for the element, neighbor, and side. The extended DG kernels can support both array variables and normal variables. Test cases and relevant documents were added in MOOSE. We could extend libMesh in the future to directly support variables defined on element sides without the need of lower-dimensional mesh blocks, but the current implementation in MOOSE is sufficient to deliver the correct residual and Jacobian evaluations required by PJFNK solvers. Mesh displacement is supported for HFEM. In the future, we need to make sure that the implementation works with the distributed mesh with domain decomposition.

2.2.2 Implementation of the HFEM-diffusion Solver

The diffusion equation in a weak form with HFEM is:

$$(\vec{\nabla}\phi^*, D\vec{\nabla}\phi)_{\mathscr{D}} + (\phi^*, \sigma_a\phi)_{\mathscr{D}} - (\chi^*, [\phi])_{\Gamma} - ([\phi^*], \chi)_{\Gamma} + \frac{1}{2}(\phi^*, \phi)_{\partial\mathscr{D}} = (\phi^*, q)_{\mathscr{D}}, \tag{10}$$

where ϕ is the scalar flux and χ is the Lagrange multiplier that is equivalent to the surface net current. The test functions corresponding to the element and surface variables are denoted as ϕ^* and χ^*, respectively, D is the diffusion coefficient, σ_a is the absorption, q is the volumetric source. $\mathscr{D}$ is the union of elements

over the domain, Γ is the union of internal surfaces, and $\partial\mathscr{D}$ is the union of surfaces on boundary. The operators $(f,g)_{\mathscr{D}}$ and $(f,g)_{\Gamma}$ or $(f,g)_{\partial\mathscr{D}}$ are the inner products of two generic variables f and g on element volume, internal surfaces, and boundary respectively, and $[f]$ represent the jump of a variable f on an internal surface. The group index is omitted, and we only considered the vacuum boundary for brevity. Compared to the weak form of the CFEM [7], the Lagrange multiplier terms defined on the internal interfaces additionally appear on the left-hand side of the equation. The HFEM weak form shares the same volumetric kernels and boundary conditions as the CFEM weak form. The net current variable was added to the element interfaces in the lower dimension mesh and the Lagrange multiplier kernels were incorporated with the net current variables. The scalar flux variables were set to use elemental variable types that are defined for individual mesh elements but not for mesh vertices. Note that the functional order of surface elements should be lower, by at least two degrees, because of the rank condition of HFEM formulation [44].

This linear system of the HFEM-Diffusion equation produced by the weak form is currently directly solved by the SuperLU-DIST package linked through PETSc [28]. The linear system is a saddle point problem due to the presence of Lagrange multiplier coupling across the element interfaces. The conventional preconditioning techniques are not adequate for solving this type of problem. Thus, the direct matrix inversion technique employing the LU decomposition is used in the HFEM-diffusion solver. Consequently, its performance is rapidly degraded with increasing number of unknowns. The performance of the HFEM solver can be significantly improved with the RB iteration scheme, which is introduced in the subsequent subsection.

2.2.3 Extension of the HFEM Solver to the PN Equation

The HFEM solver in Griffin has been extended from the diffusion equation to the P_N equation in order to properly account for the dominant transport effect of fast reactors originating from pronounced anisotropic scattering. The extension of the HFEM solver starts from the transport equation represented using even- and odd-parity angular fluxes as:

$$\vec{\Omega}\cdot\vec{\nabla}\Psi^{-}(\vec{\Omega})+\boldsymbol{H}[\Psi^{+}(\vec{\Omega})]=S^{+}(\vec{\Omega}), \tag{11}$$

$$\Psi^{-}(\vec{\Omega})=\boldsymbol{G}[S^{-}(\vec{\Omega})]-\boldsymbol{G}[\vec{\Omega}\cdot\vec{\nabla}\Psi^{+}(\vec{\Omega})], \tag{12}$$

where Ψ^+ and Ψ^- are the even- and odd-parities of angular fluxes, respectively, and corresponding source terms are denoted as S^+ and S^-. Two operators $\boldsymbol{H}$ and $\boldsymbol{G}$ appearing in the even- and odd-parity equations are defined as:

$$\boldsymbol{H}[f^+(\vec{\Omega})] = \sum_{n \text{ even},m} (\sigma_t - \sigma_{s,n}) R_{n,m}(\vec{\Omega}) \int_{\mathscr{S}} Y_{n,m}(\vec{\Omega}') f^+(\vec{\Omega}') d\vec{\Omega}', \tag{13}$$

$$\boldsymbol{G}[f^-(\vec{\Omega})] = \sum_{n \text{ odd},m} \frac{1}{\sigma_t - \sigma_{s,n}} R_{n,m}(\vec{\Omega}) \int_{\mathscr{S}} Y_{n,m}(\vec{\Omega}') f^-(\vec{\Omega}') d\vec{\Omega}', \tag{14}$$

where $Y_{n,m}(\vec{\Omega})$ is the standard spherical harmonics with double index n, m and $R_{n,m}(\vec{\Omega})$ is the paired spherical harmonics with $\int_{\mathscr{S}} R_{\ell,k} Y_{n,m} d\Omega = \delta_{\ell,n}\delta_{m,k}$, and $\mathscr{S}$ is the angular domain, the 2D unit sphere in a 3D calculation. σ_t and $\sigma_{s,n}$ are the total and scattering cross sections, respectively. The weak form of the even-parity transport equation for HFEM is:

$$\begin{aligned} &(\vec{\Omega}\cdot\vec{\nabla}\Psi^{+*}(\vec{\Omega}), \boldsymbol{G}[\vec{\Omega}\cdot\vec{\nabla}\Psi^+(\vec{\Omega})])_{\mathscr{D}} + (\Psi^{+*}(\vec{\Omega}), \boldsymbol{H}[\Psi^+(\vec{\Omega})])_{\mathscr{D}} \\ &-(\Psi^{-*}(\vec{\Omega}), \vec{\Omega}\cdot\vec{n}\,[\Psi^+(\vec{\Omega})])_\Gamma - ([\Psi^{+*}(\Omega)], \vec{\Omega}\cdot\vec{n}\,\Psi^-(\vec{\Omega}))_\Gamma + (\Psi^{+*}, |\vec{\Omega}\cdot\vec{n}|\Psi^+)_{\partial\mathscr{D}} \\ &\qquad = (\Psi^{+*}(\vec{\Omega}), S^+(\vec{\Omega}))_{\mathscr{D}} + (\vec{\Omega}\cdot\nabla\Psi^{+*}(\vec{\Omega}), \boldsymbol{G}[S^-(\vec{\Omega})])_{\mathscr{D}}. \end{aligned} \tag{15}$$

Note that the inner product is defined over the angular domain as well. Two basis functions, $g(\vec{\Omega})$ and $k(\vec{\Omega})$, are defined for the angular variable composed of even- and odd-parity terms of the spherical harmonics, respectively. These allow the introduction of the $\mathrm{P_N}$ approximation in the weak formulation. Then, the solution variable of even- and odd-parity angular moments are approximated using the $\mathrm{P_N}$ expansion forms as:

$$\Psi^+(\vec{\Omega}) = g^T(\vec{\Omega})\phi, \tag{16}$$

$$\Psi^-(\vec{\Omega}) = k^T(\vec{\Omega})\chi, \tag{17}$$

where ϕ and χ are the moment vectors consisting of expansion coefficients of even- and odd-parity spherical harmonics functions, respectively.

We note that, for $\mathrm{P_1}$ with $g(\vec{\Omega}) = R_{0,0}$ and $k(\vec{\Omega}) = R_{1,0}$ with polar direction oriented with $\vec{n}$ (i.e., $g(\vec{\Omega}) = \frac{1}{4\pi}$, $k(\vec{\Omega}) = \frac{3}{4\pi}\vec{\Omega}\cdot\vec{n}$), with isotropic external source $S = \frac{1}{4\pi}S_0$, we retrieve the diffusion weak

form Eq. (10) with

$$D = \frac{1}{3\left(\sigma_t - \sigma_{s,1}\right)}, \tag{18}$$

$$\sigma_a = \sigma_t - \sigma_{s,0}, \tag{19}$$

$$q = S_0. \tag{20}$$

The essential kernels of the HFEM-PN equation used in the evaluation of the residual and Jacobian terms were implemented in Griffin using the MOOSE array variables [45], which optimize memory usage and computational efficiency. These components of the HFEM-PN solver are assembled in the Griffin transport system framework. This work will be completed in early FY-22.

2.2.4 Red-Black Iteration

The success of HFEM-PN and HFEM-Diffusion methodologies rely on the implementation of the RB iteration. The equation with elemental even parities (scalar fluxes) and interface odd parities (currents) is recast into an equation with two partial currents on interfaces from the element to its neighbor and its neighbor to the element and on boundaries. Then we can paint the domain with a minimum number of colors so that all elements with the same color are not neighboring each other. For a regular Cartesian grid, two colors are sufficient, which is the reason why the iteration is named RB. We can assemble the response matrices for all the elements, with which we can apply the incoming partial currents to obtain the outgoing partial currents. We can update the outgoing partial currents of all elements of the same color simultaneously. Then we can iterate over the colors until the partial currents are converged. Finally, we can reconstruct the even parity fluxes in elements and the odd parity fluxes on interfaces from the converged partial currents.

We have developed the RB object for performing the RB iteration. We note that this RB object shares some similarities with the sweeper, which thus allows the two to be derived from the same base element object. Similarly to the sweeper, this RB iteration object can be used as either a preconditioner or an update operation with outer iterative schemes. We are currently in the process of adding the response matrices and testing the implementation. The benchmark section will only show the results and convergence study obtained with the PJFNK solver without the RB iteration. The performance results of HFEM-PN and HFEM-Diffusion will be added in the future.

2.3 Other Improvements in MOOSE/Griffin

Various improvements in MOOSE and Griffin were conducted to enhance the performance of Griffin. The improvements in MOOSE are supported by the NEAMS multiphysics area. These improvements demonstrated that this type of framework tasks can greatly benefit the development of individual applications, including Griffin. They comprise:

- Improvements to the new eigenvalue solver: We recently finished refactoring the new eigenvalue executioner in MOOSE (refer to MOOSE Issue #15791, #16191, #17026 #18364 #18493 in MOOSE Github repository. The major improvements include using the initial guess from the previous simulation when reentering the solver; adding colored outputs support; enabling outputting the eigenvalue at each power iteration and the end of Newton solver; separating power iteration and Newton; removing unnecessary PETSc/SLEPc options prefix; adding a new solving option that uses the pre-assembled full left-hand-side and right-hand-side matrices without performing residual evaluations during linear solves. These enhancements make the use of new eigenvalue solvers consistent with the old solvers and enable Griffin to do the NDA calculations with array variables/kernels.

- Improvements on using Google performance tools: We added a new heap profiling capability in MOOSE using Google performance tools (refer to MOOSE pull requests [PR] #16224, #16277, #16284, #16317 in the MOOSE Github repository). This, along with the CPU profiling capability, gives developers a powerful tool to pinpoint memory or CPU-time issues of MOOSE-based applications, including Griffin. This capability has been used for pinpointing the spots where code optimizations are desired.

- Memory optimization:

  - Fixed some member variables in the MOOSE assembly class that uses a lot of memory when the number of variables is large. Assembly allocates memory as the size of the square of the number of variables for these variables. In heterogeneous transport calculations, the memory usage by these variables could potentially reach one GB. We refactored the code to take advantage of the sparse coupling and thus reduce the memory usage to the order of the number of variables (refer to MOOSE Issue #16229). The memory usage is negligible with the refactored code, where we create the coupling flag matrix as a diagonal matrix instead of a full matrix.

  - Removed solution localized in SlepcSupport.C (#16147). During residual and Jacobian calculations, a global parallel solution needs to be synced to a local solution as a ghosted local solution so that we can perform finite element calculations. The *localize* function in LibMesh synced the entire global solution as a huge local solution to every single processor. This caused memory spikes and also deteriorated simulation efficiency. We updated the eigenvalue solvers to use *system.update()* that syncs the global parallel solution as a local-plus-ghost solution for every processor.
  - Avoid creating unnecessary vectors (transient solution states) and matrices (#16339). The base System classes in MOOSE should not create transient libMesh systems. Instead, we now use the non-transient systems and manage the previous solution vectors on our own so that we do not excessively create previous states that we do not need. In addition, we should avoid building matrices when using pure PJFNK.
  - Refactored the Richardson executioner. We refactored the executioner with the latest solve object system in MOOSE and were thus able to get rid of the old solution vector.

- CPU time optimization:
  - Using a pre-split mesh does not have to run mesh generators (#16218).
  - Minor ArrayKernel interface change to avoid frequent memory allocation and deallocation in array kernels (#16504 and #16360). More documentation is added for the usage of the Eigen package in array kernels. All Griffin array kernels are properly updated. For the TREAT performance benchmark, we see a 17% reduction in residual evaluation equivalent.

3. BENCHMARKS

We present three benchmarks, pertaining to a variety of reactor designs, that we use to verify the performance improvements in Griffin. These benchmarks include a thermal spectrum microreactor (Empire 2D/3D), a graphite-moderated reactor (TREAT 2D), and a sodium-cooled fast reactor (ABTR 3D). These models are part of the Griffin performance test suite and are included in the Griffin repository.

3.1 TREAT

3.1.1 Description

TREAT [46, 47] is a heterogeneous, air-cooled, graphite-moderated, graphite-reflected thermal reactor. The reactor is fueled with highly-enriched (~93%) UO_2 dispersed in graphite, with a fuel carbon-to-uranium (C/U) atomic ratio of roughly 10,000 to 1. The fuel is arranged in zircaloy-clad fuel elements, with an approximately 4-ft-long central fuel section and 2-ft-long aluminum-clad graphite reflectors above and below the fuel. The elements are approximately 4-inch $\times$ 4-inch square, with chamfered corners. The reactor core can accommodate a maximum of 361 elements, arranged in a 19 $\times$ 19 array and surrounded by a permanent graphite reflector. The experiment vehicles for sample irradiation were placed at the center of the core and typically replaced 1–4 fuel elements. Sample behavior during a transient was monitored by an ex-core system of collimated detectors called a hodoscope. A partial ("half-slotted") or full ("full-slotted") row of central fuel elements was removed to provide an unimpeded path for neutrons between the experiment and the hodoscope.

The TREAT model alone presents a challenge for the Griffin heterogeneous transport solvers with its unique radial streaming paths. It is of interests in TREAT modeling and simulations to obtain reference solutions with heterogeneous transport for verifying low-order, but more efficient, homogeneous models.

In the benchmark models, we selected two 2D models, from previous work [48], for the simulations: a simple minimum critical core (MinCC) case in Fig. 5 and a M8CAL case in Fig. 6. In the 2D MinCC case, only the core region is specified with a vacuum boundary condition. For simplicity, no control rod elements (assemblies) are defined in the case. For the 2D M8CAL case, 20 control rod elements are defined in the core, and the permanent reflector is specified as well, with a vacuum boundary condition applied to the out-surfaces of the permanent reflection region.

3.1.2 Performance Results of the M8CAL-Based Model

The numbers of elements in the two geometries are given in Table 1. A Gauss Chebyshev angular quadrature is used with 8 polar and 16 azimuthal directions per octant. The number of total streaming directions with this quadrature is $4 \times 8 \times 16 = 512$ for a 2D core. Two sets of macroscopic cross sections with 11 groups and 23 groups are used, which were generated using Serpent2 [49] from the 2D core. For simplicity, cross sections were categorized for compositions.

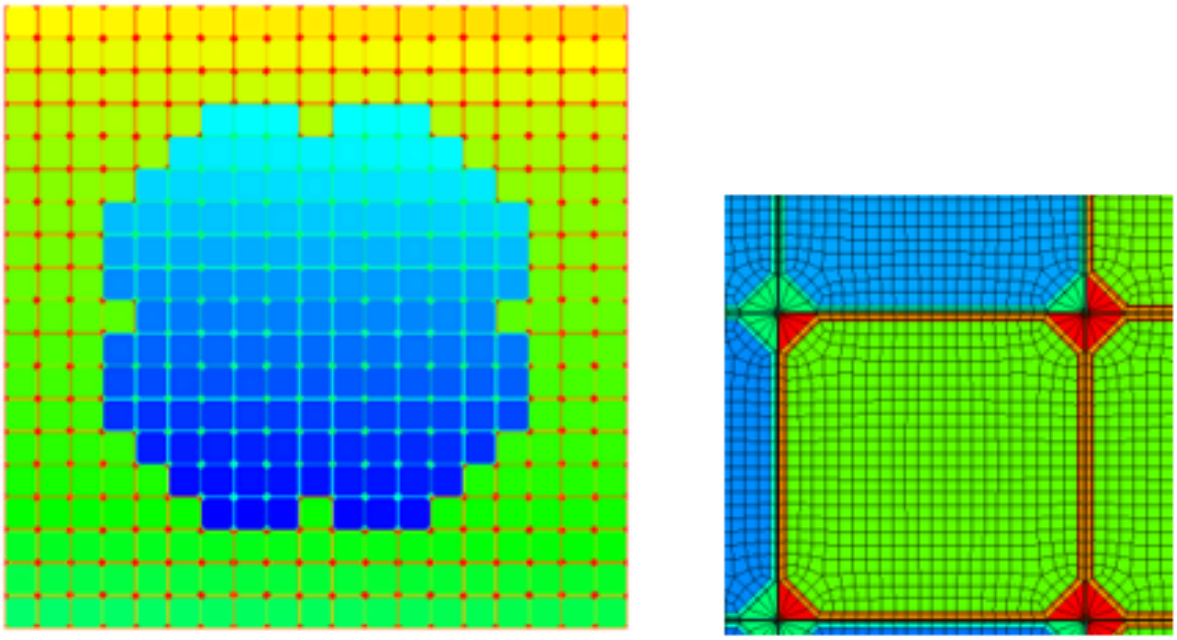

Figure 5: The MinCC-based model of the TREAT core.

Table 1: Number of Mesh Elements in the TREAT Problems.

Geometry	Mesh	# of Elements
2D MinCC	Fine mesh	251,617
	Coarse mesh	361
2D M8CAL	Fine mesh	293,485
	Coarse mesh	961

The scattering order is set to two. The CMFD accelerated Richardson iteration is used to solve this problem. Our newly developed parallel sweeper is used in the transport update. Two sweeps are employed for each transport update. The multiplicative prolongation is used. The evaluated diffusion coefficients on the coarse elements are capped with 10 cm. Four processors are used to solve the CMFD system because the number of coarse elements is small.

Calculations were performed on INL Sawtooth with 64 nodes and 12 CPUs on each node. It is noted that we do not typically use all available 48 CPUs on a node because otherwise the memory accessing bandwidth can slow down the calculation thus affect the parallel performance, a figure of merit we are interested in for benchmarking our implementations. We used the pre-split mesh for this calculation. The memory usage on the main processor of total 768 processors reported by the code after problem setup is about 714 MB, thus memory is not a constraint for running this problem on INL Sawtooth.

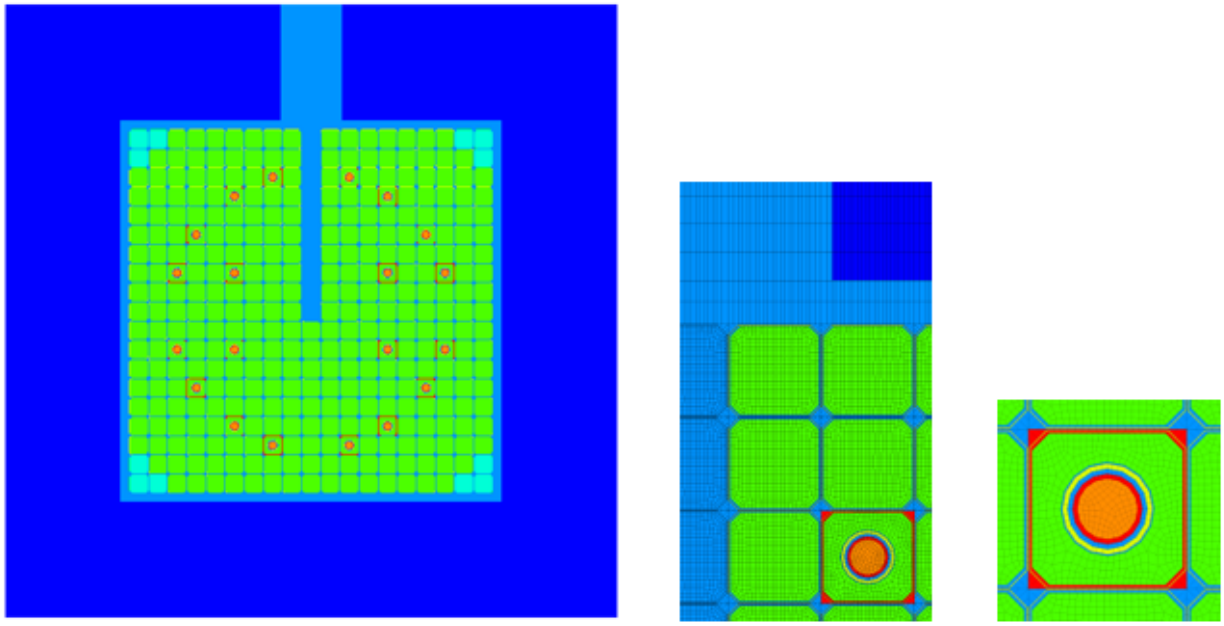

Figure 6: The M8CAL-based model of the TREAT core.

Fifteen Richardson iterations were required to converge this problem with the relative tolerance of 10^{-4} on the L2 norm of the difference of the angular flux between two successive iterations. With this tolerance, the iterative error on k-effective is less than 1 pcm. The wall time is 164.2 seconds. The problem setup time is about 2 seconds in setting up the DoF map for the angular fluxes. The majority of the CPU time can be split into the times in residual evaluation/update with sweeping/postprocessor evaluation on the transport system and the prolongation/projection/solve of the CMFD equation as in Table 2. The initialization of CMFD is negligible due to the small coarse mesh.

Table 2: Detailed CPU times for TREAT 2D benchmark with 64×12 processors with CMFD.

Name	CPU Time (s)	% CPU Time
Transport residual evaluation (15)	27.575	16.79
Transport sweeper (15×2)	108.540	66.10
Transport postprocessor evaluation (17)	2.433	1.48
CMFD projection (15)	4.116	2.51
CMFD prolongation (15)	9.878	6.02
CMFD solve (15)	5.589	3.40
Wall time	164.212	

The numbers in the parenthesis of six operations in Table 2 are how often these operations were called. Because the number of streaming directions is relatively large, 512, the CPU times in the CMFD projection/prolongation and transport residual evaluation/sweeping are large, especially much larger than the time in the CMFD solve. The grind time of residual evaluation becomes smaller with increased number of streaming directions due to the cache advantages provided by array variables/kernels [45].

For the 2D M8CAL-based TREAT benchmark, Griffin and PROTEUS-MOC were run using the same cross sections and 40 processors at the ANL Linux cluster. From a mesh convergence study, it was found that PROTEUS-MOC required a finer mesh for the 61-cm-thick permanent graphite reflector region, since the core region is almost full of fuel blocks and thus thermal fluxes are dramatically changing through the permanent reflector region, as shown in Fig. 7. Therefore, a 536,805 element mesh was used for PROTEUS-MOC to obtain the converged solution in terms of mesh refinement, while a relatively smaller number of elements (293,485) were allowed for Griffin. As previously mentioned, PROTEUS-MOC uses double the number of angles than Griffin for 2D calculations. Due to the large difference in DoFs (almost 3×) between Griffin and PROTEUS-MOC, Griffin was able to solve the 2D M8CAL problem much faster than PROTEUS, as shown in Table 3.

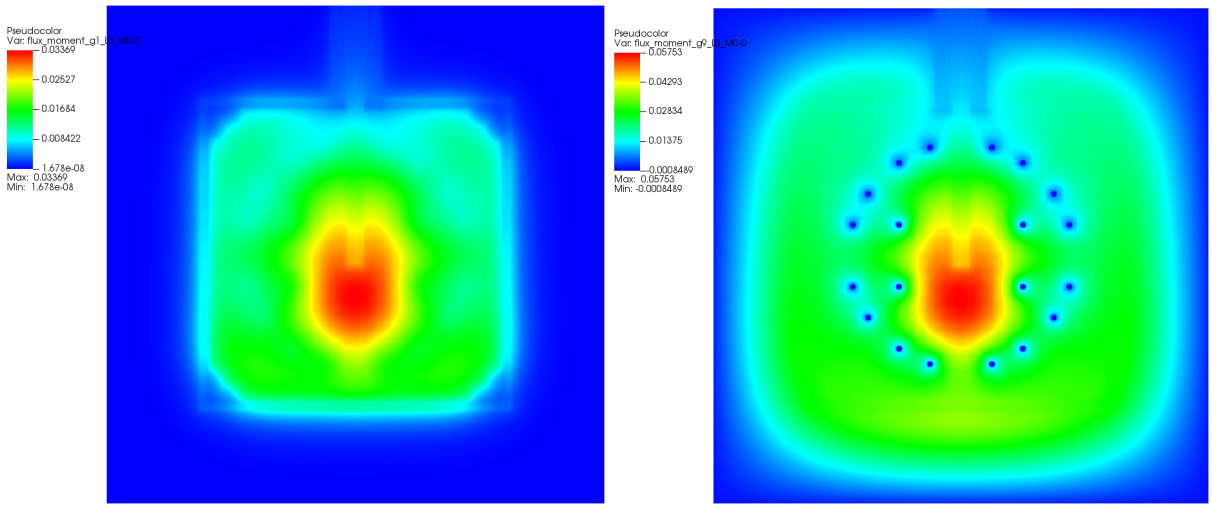

Figure 7: Fast (left) and thermal (right) flux solutions of the TREAT M8CAL-based model.

Table 3: Griffin solutions and performance for the 2D M8CAL-based TREAT benchmark.

Angle	Griffin (DFEM-SN, CMFD)			PROTEUS (MOC, CMFD) *			Reference
	k-eff	Memory	Time	k-eff	Memory	Time	k-eff
24	0.92704 (15)	37.8 G	7 m 52 s	0.92408 (11)	43.4 G	17 m 31 s	0.92746
48	0.92293 (15)	51.5 G	10 m 4 s	0.91996 (11)	68.0 G	37 m 23 s	0.92328
72	0.92238 (15)	64.3 G	14 m 40 s	0.91944 (11)	93.7 G	59 m 20 s	0.92273

* PROTEUS uses double the number of angles for 2D calculations, which almost doubles the computation time.

Used mesh elements: 293,485 for Griffin, 536,805 for PROTEUS

Used 11 group cross sections and 40 processors on the ANL Linux cluster

(#) number of iterations

The "reference" k-eff were generated with Griffin DFEM-SN using two uniform mesh refinements ($293,485 \times 16 = 4,695,760$ elements) and with second-order Lagrange shape functions, which have much smaller spatial discretization errors than the other k-eff in Table 3. Fixed 2 polar angles were employed for all calculations. The number of Richardson iterations in Griffin (15) is larger than the number of iterations in PROTEUS-MOC (11), which is due to the lower number of source iterations, thus lower computing time, in each transport update.

3.1.3 Performance Results of the MinCC-Based Model

The performance results of the MinCC-based model is similar. The number of elements is 251,617, slightly smaller than the 293,485 elements in the M8CAL-based model. The coarse grid is a regular Cartesian, with only 361 elements. It takes 10 CMFD accelerated Richardson iterations to converge, and the wall time is 90.5 seconds. The number of Richardson iterations is slightly smaller than the number of M8CAL-based iterations due to a simpler configuration of MinCC, such as no hodoscope, etc. The time distribution is shown in Table 4.

Table 4: Detailed CPU times for TREAT MinCC-based 2D benchmark with 64×12 processors with CMFD.

Name	CPU Time (s)	% CPU Time
Transport residual evaluation (10)	15.804	17.47
Transport sweeper (10×2)	59.390	65.65
Transport postprocessor evaluation (12)	1.526	1.69
CMFD projection (10)	1.492	1.65
CMFD prolongation (10)	5.773	6.38
CMFD solve (10)	1.289	1.42
Wall time	90.463	

For the TREAT MinCC-based 2D benchmark case, the eigenvalue, memory, and computation time of Griffin were compared with those of PROTEUS-MOC, for which different angles (24, 48, and 72) with fixed 2 polar angles and energy groups (11 and 23) were used. Both codes were run using 40 processors at the ANL Linux cluster (Xeon 6148 CPU, 2.40 GHz, and 512 GB memory/node). A CMFD acceleration was applied to both codes. Note that PROTEUS-MOC solves a 2D problem with the same number of angles used for a 3D problem, requiring twice the number of angles as Griffin. As shown in Table 5, the wall-clock computation times of the two codes are comparable to each other, even though Griffin needed slightly more memory for most cases. The convergence behavior with an increased number of angles is similar in both codes as well. Eigenvalue differences of about 200 pcm are observed between the two codes, which could be further reduced with mesh and angle refinement.

Table 5: Griffin solutions and performance for the 2D TREAT MinCC-based benchmark.

Group	Angle	Griffin (DFEM-SN, CMFD)			PROTEUS (MOC, CMFD) *			Reference
		k-eff	Memory	Time	k-eff	Memory	Time	k-eff
11	24	1.18835 (10)	24.0 G	3 m 32 s	1.18609 (7)	30.6 G	3 m 4 s	1.18842
	48	1.18726 (10)	44.0 G	4 m 49 s	1.18505 (6)	40.0 G	4 m 50 s	1.18737
	72	1.18674 (10)	56.3 G	6 m 12 s	1.18458 (7)	52.9 G	7 m 25 s	1.18688
23	24	1.18774 (10)	43.9 G	7 m 3 s	1.18548 (7)	42.8 G	6 m 32 s	1.18781
	48	1.18665 (10)	78.4 G	10 m 45 s	1.18443 (7)	56.1 G	9 m 24 s	1.18676
	72	1.18613 (10)	93.4 G	14 m 23 s	1.18396 (7)	71.0 G	14 m 18 s	1.18627

* PROTEUS uses double the number of angles for 2D calculations, which almost doubles the computation time

Used 40 processors at the ANL Linux cluster (Xeon 6148 CPU, 2.40 GHz, 512 GB/node)

(#) number of iterations

The "reference" k-eff were generated with Griffin DFEM-SN with two uniform mesh refinements ($251,617 \times 16 = 4,025,872$ elements) and with second-order Lagrange shape functions.

The increase rate in memory usage with the increased number of streaming directions is slightly faster for Griffin than for PROTEUS-MOC with the same mesh. We believe that this is because the memory usage per direction by Griffin is higher with the ghosted solution vectors managed through PETSc parallel vectors. We plan to improve the memory management in the MOOSE framework to address this in the future. This issue can have a larger impact for 3D problems where the required number of nodes due to the memory constraint are not available.

3.2 Empire

3.2.1 Description

Empire [50] is a microreactor specification developed under the DOE ARPA-E MEITNER program. A unit assembly is comprised of a stainless-steel monolith with 217 total holes drilled with a triangular pitch, containing 61 sodium/stainless-steel heat pipes, 96 moderator pins, and 60 fuel pins. There are air gaps between the fuel and moderator materials and the monolith; the monolith serves as the cladding for all pins. Consisting of 18 fuel assemblies surrounded by a beryllium reflector, the full core was envisioned to have

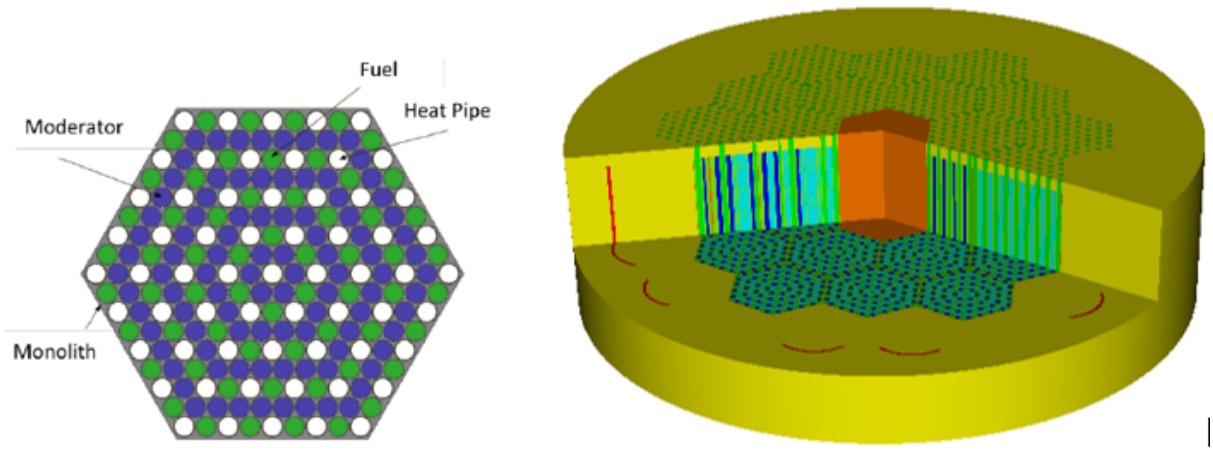

Figure 8: Fuel assembly and core configurations of the Empire microreactor.

a central void region that could accommodate a safety/shutdown rod and to have 12 control drums in the outer reflector region. In this specification, the central region is simply an assembly-size region assumed to be filled with air.

There is an assumed inter-assembly gap. However, no gap exists in the radial reflector or between the control drum block and the radial reflector; we assumed that this region would be manufactured from a monolithic beryllium unit. The core is 70-cm high, including axial reflectors, each of which is 5-cm high, and an active core of 60-cm tall. As shown in Fig. 8, the bottom reflector has no penetrations or gaps, while heat pipe holes and the center hole exist in the top reflector. This configuration naturally leads to significant neutron leakage. This leakage could be mitigated by the introduction of an additional reflector. It is observed that the magnitude of the leakage may provide a challenge to modeling efforts.

Each control drum has an accurate control absorber, with 1-cm-thick EuB_6. Only two rotations of control absorbers within a control drum are selected in the current work: all control drums turned fully in (zero rotation) and fully out (180° rotation). In both control states, the center of the control absorber is lined up between the two neighboring fuel assemblies, as shown in Reference [50]. For the performance tests here, a core model with a hexagonal boundary, as shown in Fig. 9, was developed to make it easy to create a coarse mesh for the CMFD acceleration.

The 2D mesh contains 402,444 elements with twenty-one mesh blocks. Detailed mesh information is listed (from the Griffin screen output):

```
Number of elements = 402444
   Number of nodes = 300565
   Number of sides = 703008
Blocks:
      0 (REFL) : volume 1119.137979 with 13020 elements,  Coordinate system: XYZ, Elem types: TRI3
```

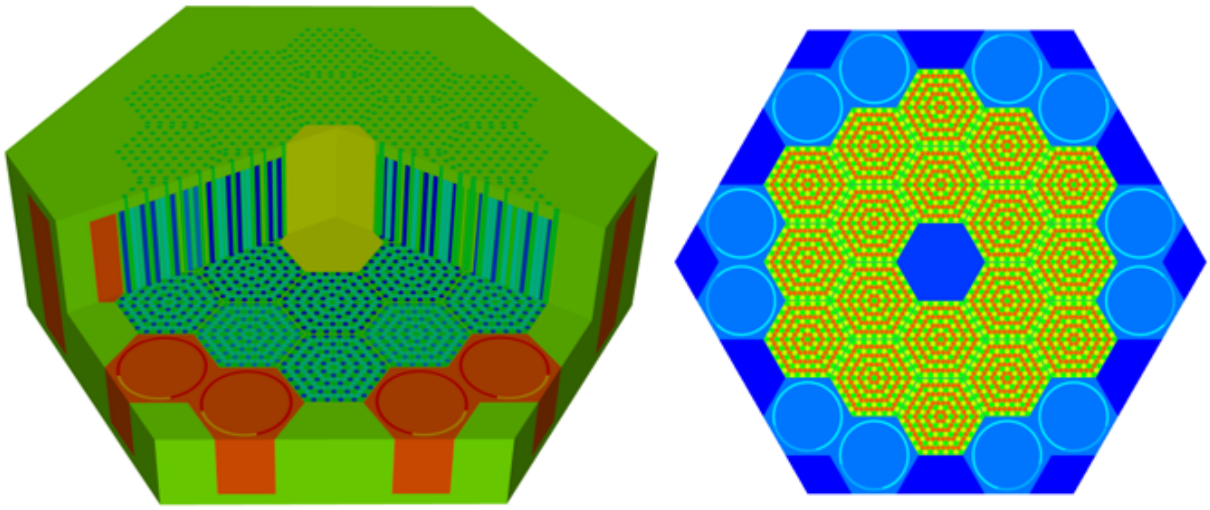

Figure 9: 3D core problems with hexagonal boundaries: 3D (left) and 2D (right) views.

```
      1 (REFL_QUAD) : volume 3357.414002 with 13020 elements,  Coordinate system: XYZ, Elem types: QUAD4
      2 (AIR) : volume 268.1463905 with 4896 elements,  Coordinate system: XYZ, Elem types: QUAD4
      3 (CRREFL) : volume 9799.306482 with 125472 elements,  Coordinate system: XYZ, Elem types: TRI3
      4 (CR) : volume 269.5849576 with 4464 elements,  Coordinate system: XYZ, Elem types: TRI3
      5 (CRREFL2) : volume 808.9066515 with 12576 elements,  Coordinate system: XYZ, Elem types: TRI3
      6 (HPIPE) : volume 383.2742216 with 13176 elements,  Coordinate system: XYZ, Elem types: TRI3
      7 (HPIPE_QUAD) : volume 3066.193998 with 26352 elements,  Coordinate system: XYZ, Elem types: QUAD4
      8 (SS) : volume 4012.052864 with 46872 elements,  Coordinate system: XYZ, Elem types: QUAD4
      9 (FUEL4) : volume 107.5209947 with 4320 elements,  Coordinate system: XYZ, Elem types: TRI3
     10 (FUEL4_QUAD) : volume 860.1680932 with 8640 elements,  Coordinate system: XYZ, Elem types: QUAD4
     11 (FGAP) : volume 322.3275167 with 12960 elements,  Coordinate system: XYZ, Elem types: QUAD4
     12 (MOD) : volume 544.3751077 with 20736 elements,  Coordinate system: XYZ, Elem types: TRI3
     13 (MOD_QUAD) : volume 4355.001767 with 41472 elements,  Coordinate system: XYZ, Elem types: QUAD4
     14 (MGAP) : volume 529.2947428 with 20736 elements,  Coordinate system: XYZ, Elem types: QUAD4
     15 (FUEL5) : volume 107.5209901 with 4320 elements,  Coordinate system: XYZ, Elem types: TRI3
     16 (FUEL5_QUAD) : volume 860.1680705 with 8640 elements,  Coordinate system: XYZ, Elem types: QUAD4
     17 (FUEL) : volume 107.5209762 with 4320 elements,  Coordinate system: XYZ, Elem types: TRI3
     18 (FUEL_QUAD) : volume 860.1680199 with 8640 elements,  Coordinate system: XYZ, Elem types: QUAD4
     19 (AIRHOLE) : volume 99.47895868 with 2604 elements,  Coordinate system: XYZ, Elem types: TRI3
     20 (AIRHOLE_QUAD) : volume 795.83145 with 5208 elements,  Coordinate system: XYZ, Elem types: QUAD4
Side sets:
      0 (SIDE_SET0000001) : volume 336.2222389 with 630 sides
      1 (SIDE_SET0000003) : volume 112.0740791 with 210 sides
      2 (side_set_0000001) : volume 224.1481588 with 408 sides
```

The 2D mesh is extruded in 3D with 17 layers in the 3D mesh. The number of coarse elements is 2,108. The extruded 3D coarse mesh contains five layers. The first fine layer corresponds to the first coarse layer. The evenly sized middle 15 layers correspond to the evenly sized middle three coarse layers. The last fine layer corresponds to the last coarse layer.

For the performance tests, we used a core model with a hexagonal boundary, as shown in Fig. 10, which includes 18 fuel assemblies, 12 control drums, and one assembly-sized hole at the center. In the coarse mesh, a fuel assembly and a control drum were meshed with 96 and 12 elements, respectively.

Table 6: Number of mesh elements in the empire problem.

Geometry	Mesh	# of Elements
2D	Fine mesh	402,444
	Coarse mesh	2,160
3D	Fine mesh	5,634,216
	Coarse mesh	10,800

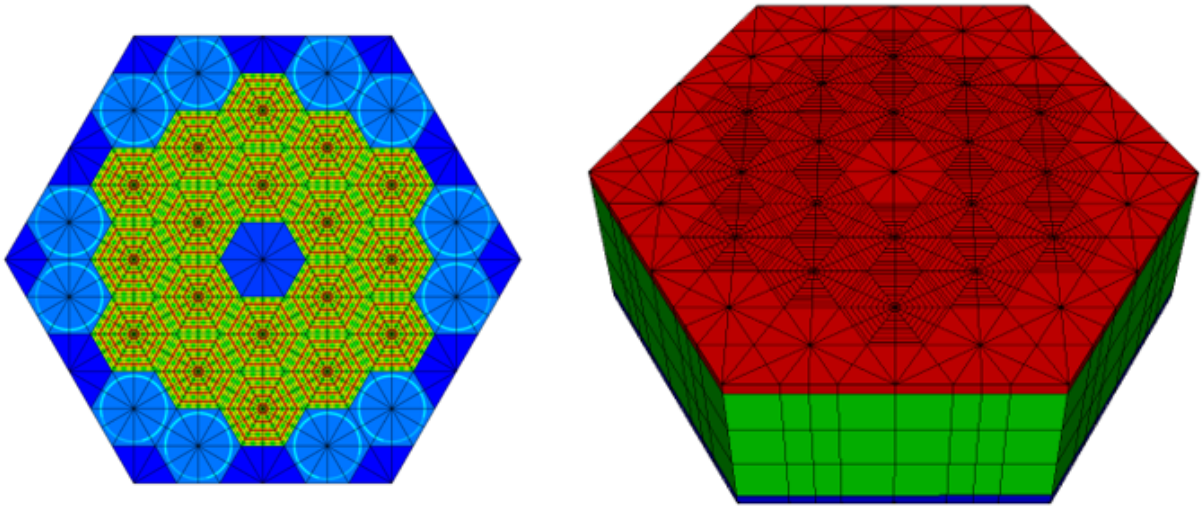

Figure 10: CMFD mesh for the full core with control drums: 2D (left) and 3D (right) mesh.

Cross sections in 11 and 23 energy groups were generated using Serpent2. Cross-section outputs from Serpent2 were processed to generate an ISOTXS format of cross sections, which were converted to the XML format using the ISOXML module in Griffin. Cross sections were simply generated for different compositions: fuel pin, moderator pin, heat pipe, air gap and hole, monolith, reflector, and control absorber.

We show the performance of the fine-mesh NDA and CMFD with DFEM-SN with the Empire 2D benchmark. The difference of the two methods is in that one uses the same fine mesh for the diffusion system and the other uses a coarse mesh. They both use the finite difference for the diffusion system, and they both perform Picard iterations between the diffusion solve and the transport update. CMFD is done with the new CMFD acceleration object, while a fine-mesh NDA is with our current NDA implementation. The fine-mesh NDA exhibits a faster convergence of the Picard iteration, but has a costly diffusion solve. CMFD requires more transport sweeps for the same speed of the Picard convergence, but the diffusion solve is extremely fast with the coarse mesh. They give the identical solution upon convergence. We also show the performance of CMFD with DFEM-SN with the Empire 3D benchmark.

3.2.2 2D Fine-Mesh NDA

The number of elements is 402,444. There are 11 energy groups. The Gauss-Chebyshev angular quadrature is employed with three polar angles and four azimuthal angles per octant, thus totaling 48 streaming directions. The scattering order is one. The diffusion system uses the same mesh as the transport system but uses a constant monomial as the shape function to have a faster solve. The diffusion system uses the matrix-only Newton solver with DSA, whose residual evaluations are through assembled matrices. Our newly developed parallel sweeper is used for the transport update. One sweep is employed for each transport update. Additive prolongation is used.

We will show CPU times with a calculation performed on Sawtooth with 64 nodes and 12 CPUs on each node. We chose 12 CPUs, not because we cannot use more CPUs, but rather wanted to show CPU times with 768 processors with a reasonable allocation of computing resources. We use the pre-split mesh for this calculation. The memory usage on the main processor reported by the code after the problem setup is about 267 MB, thus memory is not a constraint for running this problem on INL Sawtooth.

The wall time is 54.4 seconds. The problem setup time is negligible in about 1 second. The majority of the CPU time can be split into the times in residual/Jacobian evaluations and the GMRes solve/preconditioning time of the diffusion system, transport residual evaluation, transport update with sweeping and transport postprocessor evaluation.

Table 7: Detailed CPU times for Empire 2D benchmark with 64×12 processors with fine-mesh NDA.

Name	CPU Time (s)	% CPU Time
Diffusion residual evaluations (58)	14.951	27.46
Diffusion Jacobian evaluations (1)	0.640	1.18
Total Diffusion time	26.739	49.12
Prolongation/projection (11)	15.285	28.08
Transport residual evaluation (11)	3.082	5.66
Transport sweeper (11)	1.873	3.44
Transport postprocessor evaluation (11)	1.653	3.04
Wall time	54.440	

The numbers in the parenthesis of six operations in Table 7 are the number of the operations in the cal-

culation. The total diffusion time minus the time in diffusion residual/Jacobian evaluations is the time spent by the Newton solver for the diffusion system, including the preconditioning cost with Hypre BoomerAMG. Eleven diffusion iterations are taken. About 2.6 seconds in the initial evaluation are not included in the above table, along with some other minor parts. We notice that the cost on the transport side is only 7.3 seconds in total, less than about fifteen percent of the wall time. The majority of the time is spent in diffusion with 26.7 seconds plus 15.3 seconds in the projection and prolongation. This indicates that the fine-mesh NDA is a valid option due to the fast convergence of the diffusion acceleration and the low cost of transport update. However, we believe that we still have room to reduce the cost of the diffusion solve and data transfer in the future. It is noted that we indeed used the matrix-only solve option in the diffusion solve, which cuts down on the number of residual evaluations from over four hundred to 58 and makes the linear convergence faster with the fully assembled matrix. This method has great potential due to the extremely low number of transport sweeps required by the calculation.

3.2.3 2D CMFD

The number of elements is 402,444. The number of coarse elements is 2,160. There are 11 energy groups. The Gauss-Chebyshev angular quadrature is employed with three polar angles and four azimuthal angles per octant, thus totaling 48 streaming directions. The scattering order is one. Our newly developed parallel sweeper is used for the transport update. We used the solver with Richardson iterations accelerated by the CMFD acceleration object. At each Richardson iteration, we use two GMRes cycles (two sweeps of all energy groups) for transport update. The multiplicative CMFD prolongation is used. We used the PJFNK solver for the CMFD eigenvalue solve.

Calculations were performed on Sawtooth with 64 nodes. One, 3, 6, 12, and 24 CPUs per node are used to study the parallel performance. Fourteen Richardson iterations are used for converging the problem. Twelve processors are used for the CMFD solve.

The memory usage on the main processor reported by the code after loading the mesh is about 760 MB, 330 MB, 253 MB, 207 MB, and 255 MB, respectively. The memory usage by the solution vectors, etc. is about 50 MB with 24 CPUs and 532 MB with one CPU. The memory reporting may not be accurate by inspecting the process file, but we can still conclude that memory is not a constraint for running this problem on INL Sawtooth.

The wall times with one, 3, 6, 12, and 24 CPUs per node are 201.1, 78.1, 48.0, 32.3, and 26.8 seconds,

respectively. The CPU time can be split into the times mainly in the CMFD initialization, transport residual evaluation, transport update with sweeping, transport postprocessor evaluation, and CMFD projection/prolongation/solve at each Richardson iteration. As one example, CPU times for Empire 2D benchmark with 64×12 CPUs are presented in Table 8.

Table 8: Detailed CPU times for Empire 2D benchmark with 64×12 processors with CMFD.

Name	CPU Time (s)	% CPU Time
Transport residual evaluation (14)	4.323	13.40
Transport sweeper (14×2)	13.915	43.15
Transport postprocessor evaluation (16)	0.735	2.28
CMFD projection (14)	1.765	5.47
CMFD prolongation (14)	1.934	6.00
CMFD solve (14)	7.274	22.55
CMFD initialization (1)	0.402	1.25
Wall time	32.251	

To show the parallel scalability, we plot the parallel efficiency of five runs for transport residual evaluations, sweepings, CMFD projections and prolongations. The efficiencies are normalized to the calculation with 64 processors. The CPU time for transport postprocessor evaluation is scalable as the residual evaluation and its contribution to the total CPU time is small, and thus we do not include it in the plot.

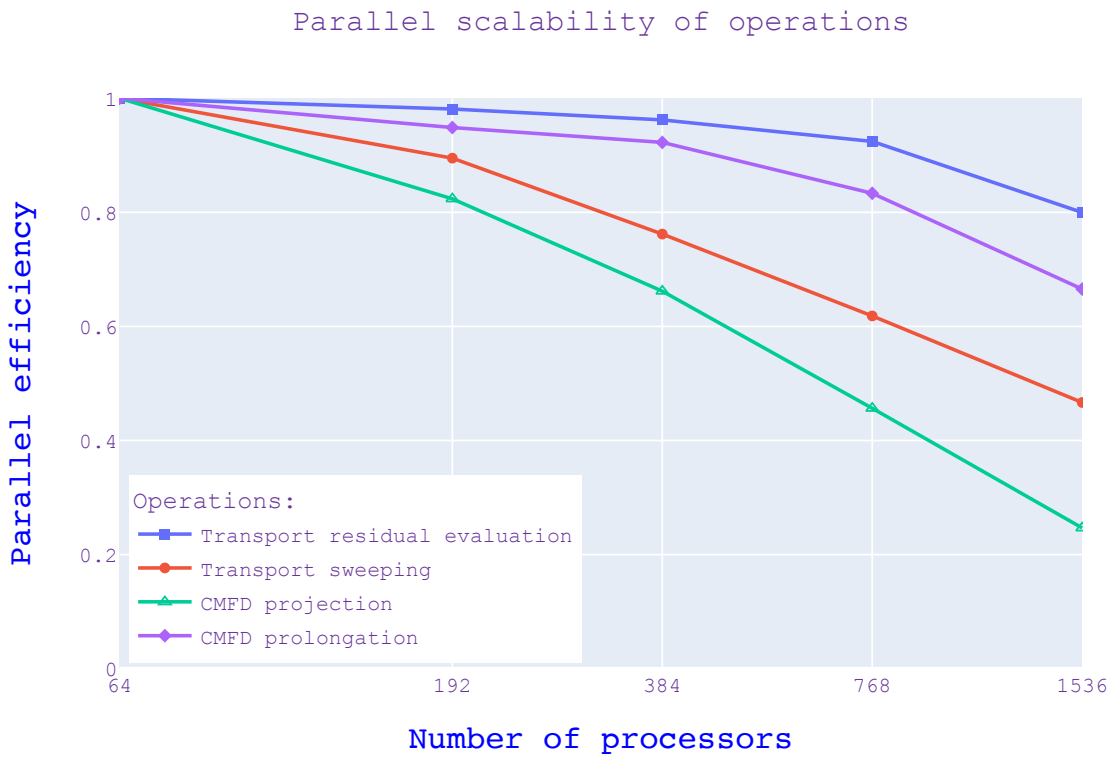

Figure 11: Parallel efficiency of various operations with Empire 2D benchmark.

We can see a good time reduction even with 1,536 processors, about 262 elements for each processor. The domain decomposition with 64×32 processors is illustrated in Fig. 12.

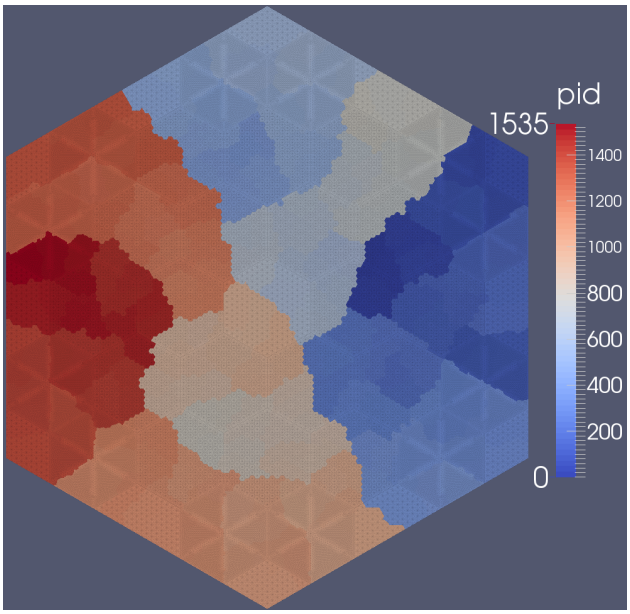

Figure 12: Domain decomposition of the Empire 2D mesh.

The parallel efficiency for transport residual evaluation is above 80%. The parallel efficiency for CMFD prolongation is not as good due to the gathering and broadcast operation from participating processors for solving all processors. The parallel efficiency for transport sweeping is not as good but is still acceptable considering the small number of elements on each processor and the irregular domain partitioning. The poor scalability of the CMFD projection operation is probably due to the gathering operation of quantities of all coarse elements and can be improved with the implementation of a distributed CMFD solve in the future.

The grind time for the transport residual evaluation with 64 processors is 0.34 microseconds (47.950 seconds divided by 14 Richardson iterations and 637,471,296 total number of degrees of freedom multiplied by 64 processors). We note that this grind time can increase with more complicated multiphysics models. The grind time for the transport sweeping is 0.37 microseconds (103.251 seconds divided by 14 Richardson iterations and two sweeps for the iteration and 637,471,296 total number of degrees of freedom multiplied by 64 processors). There could be room to further reduce the grind time of the transport sweep since we currently assemble elemental matrices for all fine elements of the arbitrary unstructured mesh. This grind time is not as good as highly optimized standalone transport sweepers but it renders a wall time meeting our requirements of most reactor analysis tasks as indicated by Empire 2D benchmark.

The Griffin eigenvalue, memory, and computation time were compared with those of PROTEUS-MOC for the 2D Empire benchmark case for various numbers of angles (24, 48, and 72) with fixed 2 polar an-

gles and energy groups (11 and 23). Both codes were run using 40 processors at the ANL Linux cluster. A CMFD acceleration was applied to both codes. As shown in Table 9, the wall-clock computation times and required memory of the two codes are comparable. The convergence behavior with an increased number of angles appear to be similar in both codes. "Reference" k-eff were obtained with Griffin DFEM-SN with two uniform mesh refinements (6,439,104 elements) and with the second order Lagrange shape functions. Eigenvalue differences of about 150 pcm are observed between the two codes, which could be further reduced with mesh and angle refinement.

Table 9: Griffin solutions and performance for the 2D Empire benchmark.

Group	Angle	Griffin (DFEM-SN, CMFD)			PROTEUS (MOC, CMFD)*			Reference
		k-eff	Memory	Time	k-eff	Memory	Time	k-eff
11	24	1.19836 (15)	33.3 G	7m 10s	1.20038 (8)	31.6 G	7m 42s	1.19775
	48	1.20000 (15)	56.9 G	11m 16s	1.20137 (8)	63.7 G	13m 44s	1.19854
	72	1.19988 (15)	81.9 G	14m 35s	1.20134 (8)	83.2 G	19m 31s	1.19865
23	24	1.19803 (14)	77.9 G	18m 29s	1.20009 (13)	68.4 G	21m 3s	1.19739
	48	1.19974 (14)	116.1 G	25m 45s	1.20113 (13)	90.9 G	35m 50s	1.19823
	72	1.19961 (14)	145.3 G	41m 12s	1.20111 (13)	112.6 G	62m 4s	1.19833

* PROTEUS uses double the number of angles for 2D calculations, which almost doubles the computation time.
Used 40 processors at the ANL Linux cluster (Xeon 6148 CPU, 2.40 GHz, 512 GB/node)
(#) number of iterations

3.2.4 3D CMFD

The number of elements is 6,841,548. The number of coarse elements is 10,800. There are 11 energy groups. The Gauss-Chebyshev angular quadrature is employed with two polar angles and three azimuthal angles per octant, thus totaling 48 streaming directions. The scattering order is one. Our newly developed parallel sweeper is used for the transport update. We used the solver with Richardson iterations accelerated by the CMFD acceleration object. At each Richardson iteration, we use eight GMRes cycles (eight sweeps of all energy groups) for the transport update. The multiplicative CMFD prolongation is used. We used the PJFNK solver for the CMFD eigenvalue solve.

Calculations were performed on Sawtooth with 64 nodes. Three, 6, 12, and 24 CPUs are used on each node to study the parallel performance. All these calculations are performed with a pre-split mesh.

The memory usage on the main processor reported by the code after problem setup are: 8,506, 4,924, 2,750, 1,627 MB, respectively. We can see that the memory parallel overhead is quite small. On Sawtooth, each CPU on average can have 4 GB maximum memory. Considering the memory used by operating system

and supporting software, we typically do not want the memory usage per CPU to be larger than 3 GB. Thus the minimum nodes for running the 3D Empire benchmark can be estimated as 1629 MB $\times$ 64 $\times$ 24/(3 $\times$ 48) GB, about 17 nodes. Because there are some memory consumption in sweepers and CMFD objects after problem setup, it is safer to say we will be needing 18 nodes of Sawtooth to run this problem. We note that further optimizations on memory usage are possible but not justified at the moment.

The wall times with three, 6, 12, and 24 CPUs per node are 108.0, 58.5, 32.4, 19.1 minutes, respectively. The problem setup time is about several seconds and is thus negligibly small compared to the wall time because we are using pre-split mesh. Some other operations, like applying initial conditions, etc., show up in the performance graph, but they are also typically in seconds and are thus negligible. The majority of CPU time (over 99%) can be split into the times in transport residual evaluation, transport update with sweeping, transport postprocessor evaluation, and CMFD projection/prolongation/solve at each Richardson iteration. As one example, with 24 CPUs per node, we have the CPU times in Table 10 with 14 Richardson iterations. The CMFD solve is done with 60 processors.

Table 10: Detailed CPU times for Empire 3D benchmark with 64×24 processors.

Name	CPU Time (s)	% CPU Time
Transport residual evaluation (14)	179.613	15.69
Transport sweeper (14×8)	721.836	63.07
Transport postprocessor evaluation (16)	16.899	1.48
CMFD projection (14)	32.599	2.85
CMFD prolongation (14)	44.029	3.85
CMFD solve (14)	126.358	11.04
CMFD initialization (1)	4.453	0.39
Wall time	1,144.539	

To show the parallel scalability, we plot the parallel efficiency of four runs for transport residual evaluations, sweepings, CMFD projections and prolongations. The efficiencies are normalized to the calculation with 64 $\times$ 3 processors. The CPU time for the transport postprocessor evaluation is scalable as a residual evaluation, and its contribution to the total CPU time is small; thus we do not include it in the plot. We can see a good efficiency for all four operations up to 1,536 processors (about 80%).

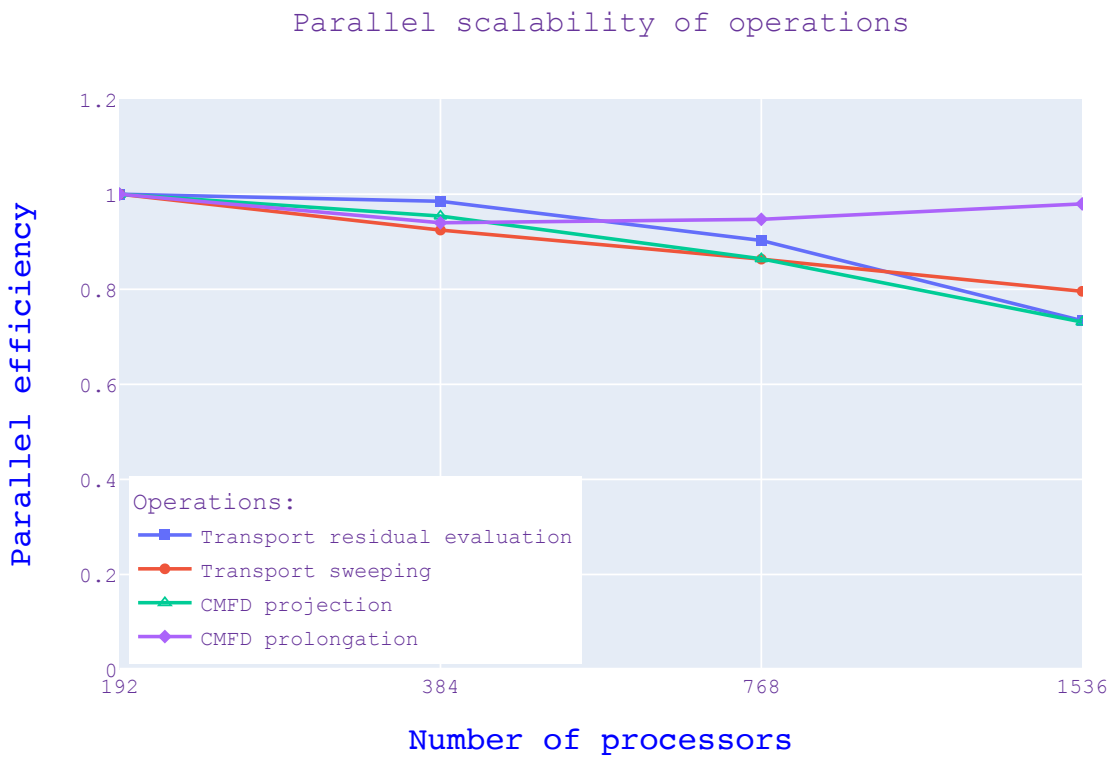

Figure 13: Parallel efficiency of various operations with the Empire 3D benchmark.

The grind time for the transport residual evaluation is 179.6 seconds divided by 14 Richardson iterations and 25,218,395,136 total degrees of freedom multiplied by 64 $\times$ 24 processors, 0.78 microseconds. We note that this grind time can increase with more complicated multiphysics models. The grid time for transport sweeping is equal to 721.8 seconds divided by 14 Richardson iterations and eight sweepers for iteration and 25,218,395,136 total number of degrees of freedom multiplied by 64 $\times$ 24 processors, 0.39 microseconds. There could be room to further reduce the grind time of transport sweeping because we currently assemble elemental matrices for all fine elements of arbitrary unstructured meshes.

3.3 ABTR

3.3.1 Description

The ABTR core design, a fast spectrum reactor shown in Fig. 14, has 199 assemblies with 14.685-cm pitch arranged in a hexagonal grid with nine rings. The core contains fuel, control, shield, and reflector assemblies with an axial height of 345.68 cm. The active core is located between 110.54 cm and 194.95 cm axially. The above core load pad is assumed to be located at 200-cm elevation.

Multigroup cross sections with typical 9 and 33 group structures were generated using MC^2-3, which were converted to the XML format for use in Griffin. The same cross-section sets were used for Griffin and PROTEUS calculations.

The meshes for the 3D ABTR benchmark cases with homogeneous or duct-heterogeneous geometry were generated using the Argonne mesh tool, based on the fully heterogeneous geometry core shown in Fig. 14. Specially for the duct-heterogeneous geometry case, the region inside a duct at each assembly was homogenized. The 3D benchmark mesh includes 19 axial nodes of about 17 cm each. For the CMFD acceleration, a coarse mesh was created with 12 elements, an assembly radially, and four total nodes axially.

3.3.2 DFEM-SN Performance for the Duct-Heterogeneous ABTR Model

There are total 190,152 elements in 156 mesh blocks. Two types of elements, HEX8 and PRISM6, are included in the 3D mesh. The mesh has 14 boundary side sets. The DFEM-SN scheme is employed for discretizing the problem. First-order monomial is used for spatial representation within the elements. Gauss-Chebyshev angular quadrature with two polar angles and three azimuthal angles per octant is used. With these settings, the total number of unknowns (angular flux degrees of freedom) is 1,204,803,072 for

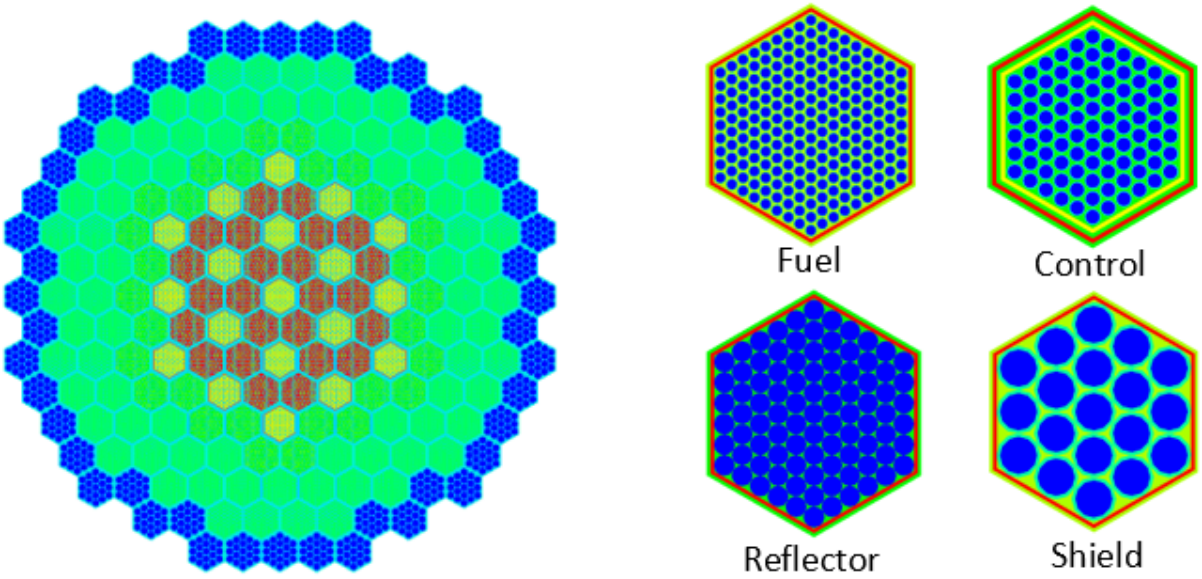

Figure 14: A heterogeneous model of the ABTR core.

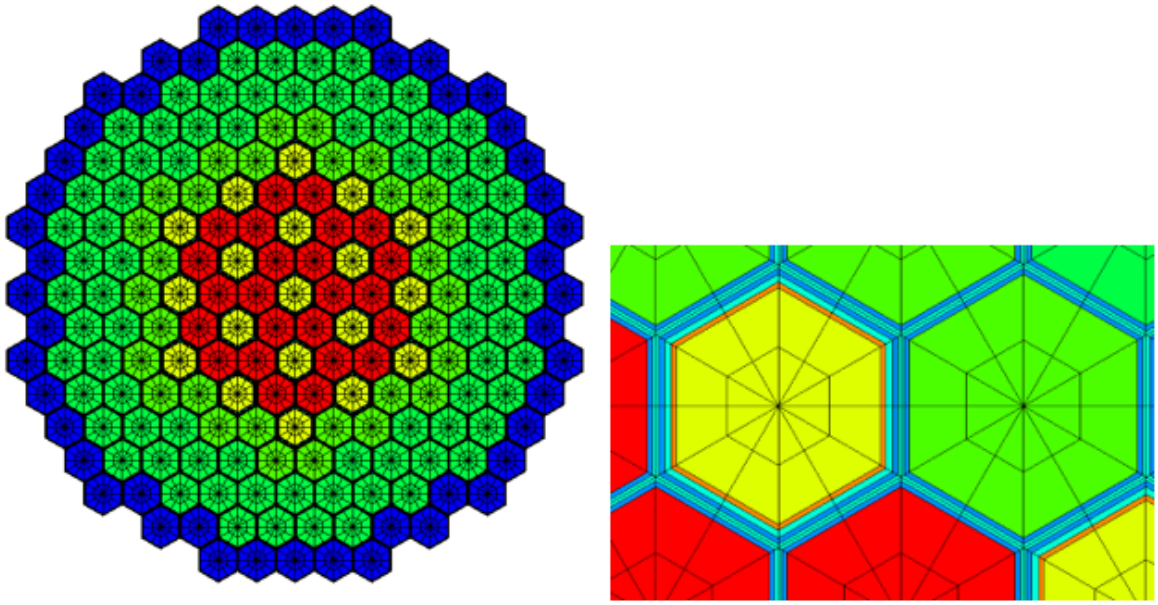

Figure 15: A duct-heterogeneous model of the ABTR core.

the 33-group case and 328,582,656 for the 9-group. Scattering order is set to be two.

We use the CMFD accelerated Richardson iteration for solving this problem. The transport update is done with the asynchronous parallel sweeper. Multiple sweeps are performed to better resolve the scattering. For the 3D problem, we use sixteen GMRes cycles for a transport update, and for 2D problem, we use eight cycles. The problem is converged so that the flux errors between two Richardson iterations is less than 10^{-5}, which also results in the error in k-effective between two Richardson iterations less than 1 pcm. For the CMFD solve, we use the distributed Krylov-Schur solver in SLEPc. Multiplicative prolongation is turned on. Pre-split meshes are used. We note that the memory usage is not a constraint for running this problem on INL Sawtooth machine. We choose to use 64 nodes on Sawtooth, with 12 CPUs on each. We summarize the CPU times in Table 11 and Table 12.

Table 11: Detailed CPU times for the 33-group ABTR benchmark with 64×12 processors.

Name	CPU Time (s)	% CPU Time
Transport residual evaluation (12)	27.348	11.68
Transport sweeper (12×16)	101.512	43.35
Transport postprocessor evaluation (14)	5.818	2.48
CMFD projection (12)	17.828	7.61
CMFD prolongation (12)	10.328	4.41
CMFD solve (12)	64.034	27.34
CMFD initialization (1)	1.982	0.85
Wall time	234.171	

Table 12: Detailed CPU times for 9-group ABTR benchmark with 64×12 processors.

Name	CPU Time (s)	% CPU Time
Transport residual evaluation (13)	6.584	14.03
Transport sweeper (13×8)	16.436	35.03
Transport postprocessor evaluation (15)	1.219	2.60
CMFD projection (13)	5.147	10.97
CMFD prolongation (13)	4.792	10.21
CMFD solve (13)	8.452	18.02
CMFD initialization (1)	1.971	4.20
Wall time	46.915	

158 processors are used in the 33-group CMFD solve and 43 processors are used in the nine-group CMFD solve. Twelve Richardson iterations are used for solving the 33-group problem and 13 Richardson iterations are used for solving the nine-group problem. The difference between the summation of all items in Table 11 and in Table 12 and the wall time is contributed with all other calculations negligible in time.

Prior to the performance tests with the 3D ABTR benchmark with duct-heterogeneous geometry, a 2D ABTR case was created to investigate the code performance in terms of mesh refinement. A base mesh was first constructed with 10,008 elements, which was refined further 4, 16, and 64 times using the Griffin's mesh refinement option. Griffin (DFEM-SN with CMFD) and PROTEUS (CFEM 2nd-order SN with DSA) were run using the same cross sections (9 groups), same meshes, and equivalent angular order (48 angles). Unlike the previous two benchmark cases which are thermal-spectrum reactors with heterogeneous geometry, PROTEUS-SN was used as a comparison code for this fast-spectrum reactor case with partially heterogeneous geometry because the heterogeneity effect is much smaller in fast reactors than in thermal reactors and the SN method performs better than the MOC for the cases where the heterogeneity effect is not dominant.

Fig. 16 shows the eigenvalue convergence behavior with mesh refinement, indicating that the eigenvalues of the two codes converged to almost the same value and that the Griffin solutions converged significantly faster than PROTEUS-SN with refined meshes by about $16\times$. The faster solution convergence of Griffin is attributed to the out-performance of DFEM over CFEM.

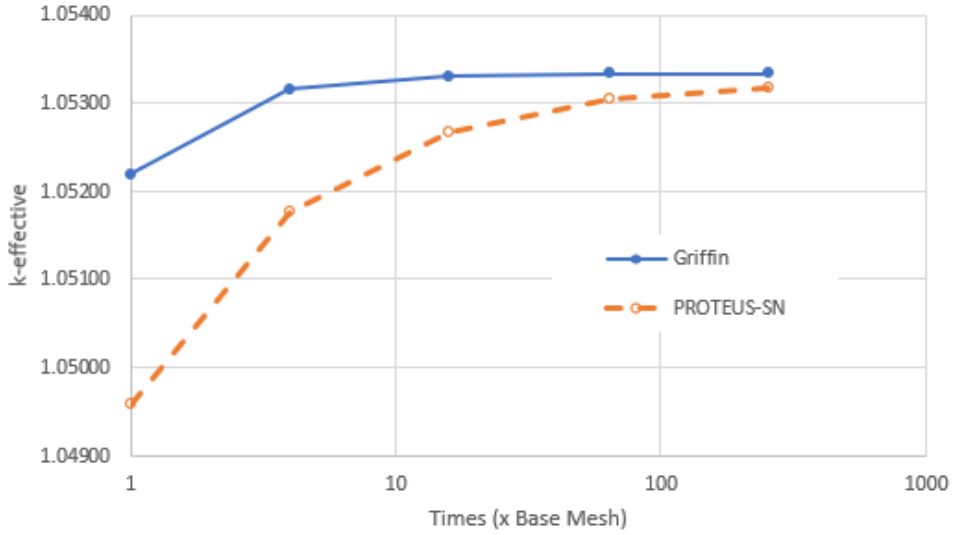

Figure 16: Eigenvalue convergence with the mesh refinement study using the 2D ABTR benchmark case.

Eigenvalue, memory, and Griffin's computation time were compared with those of PROTEUS-SN for the 3D ABTR benchmark case with duct-heterogeneous geometry, for which nine-group cross sections and 96 angles were used with three different mesh elements. Both codes were run using 40 processors at the ANL Linux cluster. A CMFD acceleration was applied to Griffin, while a DSA was used with PROTEUS-SN. Note that PROTEUS-SN uses half the number of angles used by Griffin, since it is based on the second-order even-parity SN. As shown in Table 13, the convergence behavior with an increased number of mesh elements was much better in Griffin with DFEM-SN.

Calculations were performed with differently refined meshes to compare the code performances in terms of mesh refinement since both codes basically use SN methods. The base mesh (190,152 elements) was developed using the Argonne mesh tool as shown in Fig. 15, which was further refined both radially and axially. Since it was observed that the radial refinement is more effective for the case, the mesh with 1,034,880 elements was created by refining only the one with 517,440 elements radially.

As shown in Table 13, with given angles and cross sections, eigenvalues were converged more rapidly in Griffin calculations than in PROTEUS-SN calculations. While Griffin eigenvalues were almost converged with 1,034,880 elements, PROTEUS-SN needs significantly more mesh refinement to achieve a comparable level convergence. This is mainly attributed to higher order spatial convergence for DFEM-SN with the same mesh.

Table 13: Griffin solutions and performance for the 3D ABTR benchmark with duct-heterogeneous geometry.

Mesh	Griffin (DFEM-SN, CMFD)			PROTEUS (SN, DSA)*		
	k-eff	Memory	Time	k-eff	Memory	Time
190,152	0.90876 (13)	74.7 G	20 m 55 s	0.90307 (13)	40.5 G	2 m 48 s
517,440	0.91020 (14)	186.2 G	54 m 16 s	0.90672 (13)	104.9 G	6 m 52 s
1,034,880	0.91056 (14)	350.6 G	99 m 41 s	0.90815 (13)	205.6 G	12 m 56 s
1,995,552	-	-	-	0.90845 (13)	403.4 G	28 m 48 s

* PROTEUS uses half the number of angles for 2D calculations, which almost halves the computation time.

Used 40 processors at the ANL Linux cluster (Xeon 6148 CPU, 2.40 GHz, 512 GB/node)

Used nine-group cross sections and 96 angles; (#) number of iterations

The CPU times for Griffin DFEM-SN is about 10× slower than the times with PROTEUS-SN on the same mesh and memory usage is about 1.8× higher, which is possibly an indicator for room for further optimizations in Griffin DFEM-SN. For example, we can make the grind time of the sweeper smaller for specific element types or shape function types, etc. and remove an auxiliary ghosted solution vector for residual evaluations. We can also make the CMFD diffusion solve faster. We note that the number of unknowns with DFEM-SN is equal to the number of elements (190,152 in this case) times the number of DoFs per elements (4), which is significantly larger than the number of nodes (178,340) used by PROTEUS-SN (SN2ND) with the same mesh, so the computing cost per unknown is closer for the two. Lastly and the most importantly, DFEM-SN delivers solutions with an equivalent accuracy to SN2ND with much smaller meshes. The k-eff with 190,152 elements for DFEM-SN is almost as good as the one with 1,995,552 elements for SN2ND. This confirms that the performance, based on spatial convergence, of the Griffin DFEM-SN solver is superior to that of PROTEUS-SN for the 3-D duct-heterogeneous problem.

3.3.3 Performance Test using HFEM-Diffusion

We used the ABTR homogeneous model to test the HFEM-diffusion solver. In this calculation, two-group cross sections generated from the MC^2-3 code [8] were used instead of the original 33-group cross-section set. The current solution scheme for HFEM that directly inverts the linear system with LU decomposition is not efficient for the multi-group case. The overall computational efficiency of the HFEM solver

was not covered here since it will be thoroughly investigated once the red-black iteration scheme is in place. We expect that HFEM solvers would become a valid option in terms of computational efficiency by employing the parallel RB iteration that will be available in the next fiscal year. The ABTR core model uses homogenized hexagonal assemblies in a nine-ring lattice configuration. Each assembly is divided into two quadrilateral elements since libMesh does not support a hexagonal element. The HFEM calculations were performed using various combinations of polynomial orders for volumetric and surface elements, and the obtained results are summarized in Table 14. In a separate CFEM calculation, we refined each hexagonal assembly with 128 quadrilateral elements in a radial direction and obtained a k-eff, 1.01259, as the reference for quantifying the discretization errors of the HFEM calculations. As shown in Table 14, the eigenvalues converged to the reference CFEM value to within 20 pcm with p-refinement without additional mesh refinements. Fig. 17 compares the intra-nodal flux distributions obtained using the HFEM and CFEM solvers. We confirmed that an accurate flux distribution within assemblies could be obtained by using a coarse element structure. These results suggest an advantage of higher order descriptions of solutions in design calculations based on a homogenized assembly.

The parallel performance of the HFEM solver was investigated by measuring CPU times in the residual evaluations. We did not look into the wall times because they can potentially mislead the HFEM performance due to the excessive computational burden of LU decomposition. The 2D ABTR benchmark problems were solved using one and 16 processors for the cases with 26,982 and 108,612 DOFs, respectively, and the computational times are summarized in Fig. 18 and Fig. 19. The computational time of the residual evaluation progressively decreased as the number of processors increased, thus the solver shows good parallel scalability.

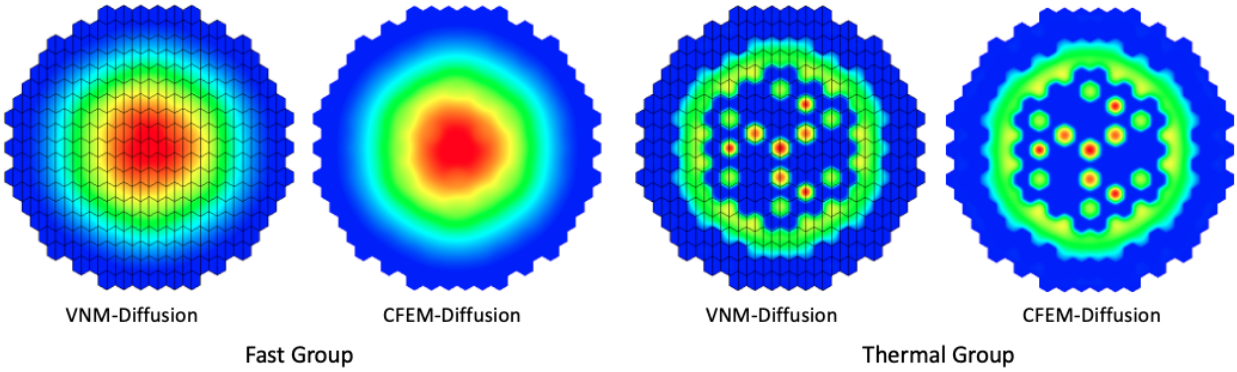

Figure 17: Comparison of flux distributions for the 3D ABTR benchmark problem.

Table 14: Eigenvalue results for 3D ABTR benchmark problem.

Basis Function Refinement		Eigenvalue	Δk^a, pcm
Volume	Surface		
1	0	0.99900	1,359
2	0	1.01652	393
3	1	1.01306	47
4	1	1.01289	30
4	2	1.01274	15

[a]Reference eigenvalue from CFEM-diffusion with an ultra-fine mesh: 1.01259

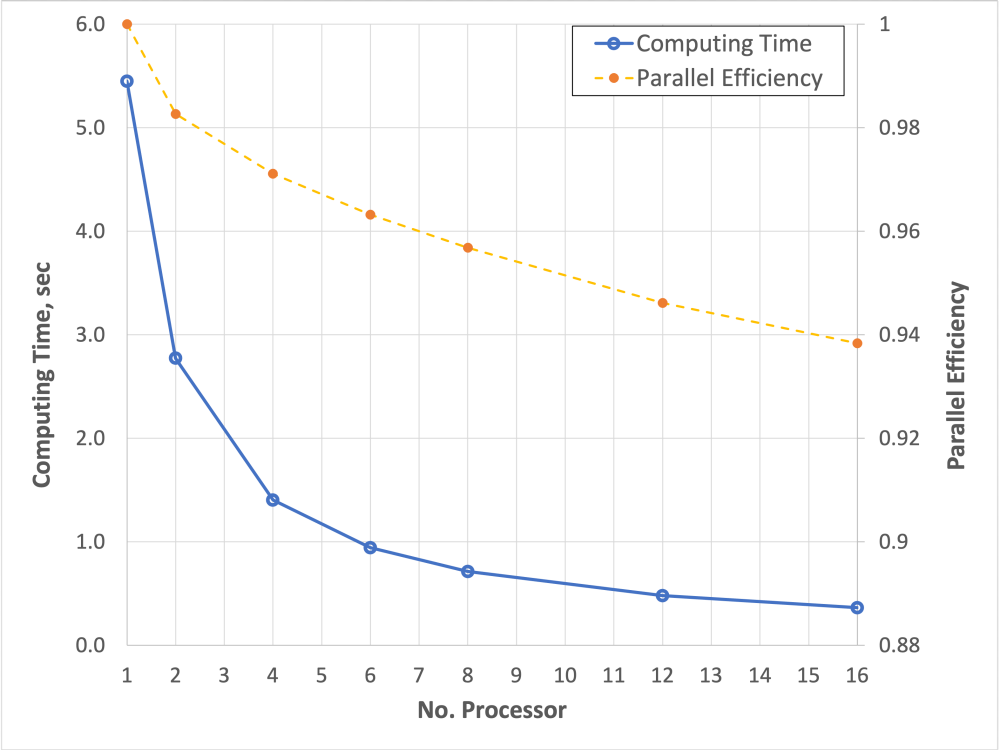

Figure 18: The parallel performance of residual evaluation in HFEM solver for 2D ABTR problem with 26,982 DOFs.

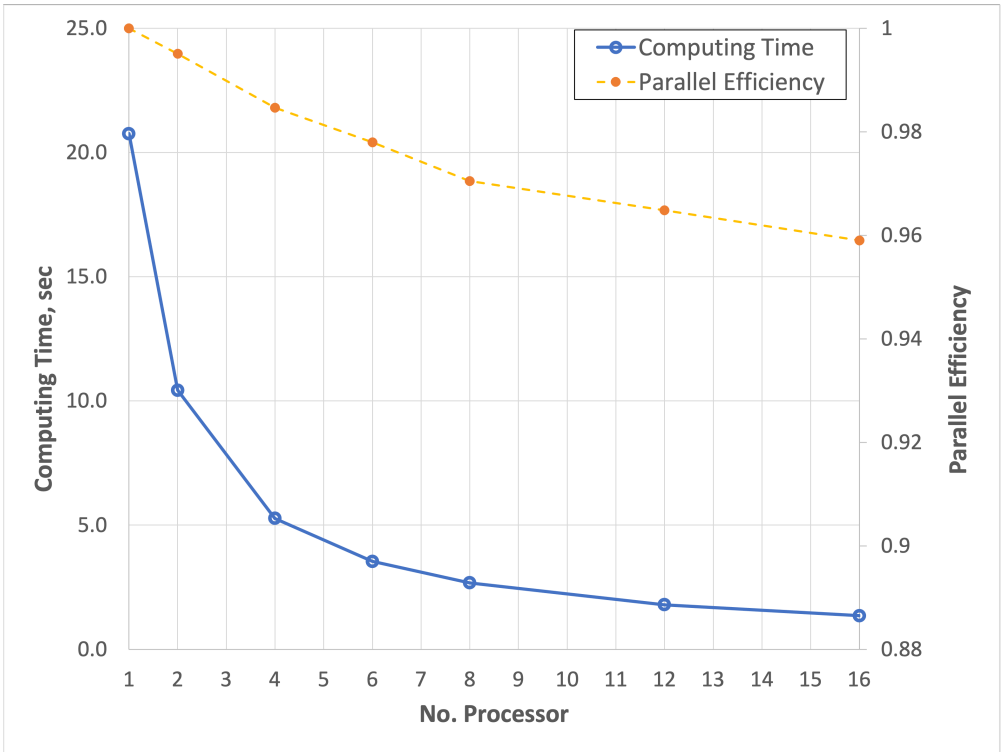

Figure 19: The parallel performance of residual evaluation in HFEM solver for 2D ABTR problem with 108,612 DOFs.

4. CONCLUSIONS AND FUTURE WORK

The Griffin DFEM-SN solver was significantly improved to support the heterogeneous transport calculations of advanced non-LWR designs. This report notes two major accomplishments: the development of the asynchronous parallel sweeper that enables an excellent parallel scalability on standard partitions of unstructured meshes, and the implementation of a CMFD acceleration object that can be seamlessly integrated into the multiphysics environment. The effectiveness of the CMFD accelerated Richardson iteration was also made more robust by investigating the multiplicative prolongation and including scattering in the transport update. With these improvements, the Griffin DFEM-SN solver can perform well for various reactor problems, as demonstrated with the TREAT, Empire, and duct-heterogeneous ABTR benchmarks.

A new solver, akin to the variational nodal method, was implemented in Griffin. This solver is based on HFEM-PN discretization and addresses the need for simulating problems with significant spatial homogenization but with the pronounced streaming inherent in fast or gas-cooled reactor systems. In addition, the HFEM-PN solver provides significant performance benefits over the previous CFEM-PN method in Griffin. We demonstrated the good parallel performance of the residual evaluation of this new solver and showed its exponential convergence with p-refinement. Lessons learned from the implementation of the DFEM-SN sweeper can be transferred to the implementation of the red-black iteration, which is a cornerstone for a high-performing HFEM-PN solver. We emphasize that, although the weak formulation and the solution techniques for the HFEM-PN method can be found in literature, our newly developed HFEM-PN is unique in the sense that it directly enables the multiphysics coupling of various feedback mechanisms.

Several areas of near-term, but future, work were identified during the performance of this research, which include:

- Making the CMFD solver support DSA/source calculations and investigate its parallel performance further
- Constructing the response matrices in the red-black iteration framework to fully demonstrate the capability of the HFEM-PN solver
- Improving the DFEM diffusion solver to reduce CPU time during CMFD accelerated transport calculations or standalone diffusion calculations

- Developing a parallel cycle detection algorithm and investigating a further reduction of grind time for the parallel sweeper
- Investigating ways for further reducing the grind time of the parallel transport sweeper
- Optimizing the memory usage related to solution vectors in MOOSE.

Mid-term, but future, tasks would be:

- Investigating the possibility of implementing a DSA preconditioner as an alternative to the Richardson iteration for SN solvers, which can possibly benefit from our improvements on the DFEM diffusion solver
- Implementing S2A (S2 transport acceleration) to address more challenging problems where CMFD may lose effectiveness
- Improving the solvers for fine-group (hundreds of groups) with and without curvilinear coordinates to support on-the-fly self-shielding treatments.

We note that all of this work will be continuously conducted within the multiphysics environment with our NQA-1 consistent SQA procedure for advanced reactor analysis, including criticality safety, depletion, design optimization, reduced-order models for operation, etc.

REFERENCES

[1] C. J. Permann, D. R. Gaston, D. Andrš, R. W. Carlsen, F. Kong, A. D. Lindsay, J. M. Miller, J. W. Peterson, A. E. Slaughter, R. H. Stogner, and R. C. Martineau, "MOOSE: Enabling massively parallel multiphysics simulation," *SoftwareX*, vol. 11, p. 100430, 2020.

[2] D. R. Gaston, C. J. Permann, J. W. Peterson, A. E. Slaughter, D. Andrs, Y. Wang, M. P. Short, D. M. Perez, M. R. Tonks, J. Ortensi, L. Zou, and R. C. Martineau, "Physics-based multiscale coupling for full core nuclear reactor simulation," *Annals of Nuclear Energy*, vol. 84, pp. 45–54, 2015.

[3] C. H. Lee and Y.S. Jung and H. Park and E.R. Shemon and J. Ortensi and Y. Wang and V.M. Labouré and Z. Prince, "Griffin Software Development Plan," Research Report INL/EXT-21-63185, ANL/NSE-21/23, Idaho National Laboratory, Argonne National Laboratory, June 2021.

[4] "SQAP for MOOSE and MOOSE-Based Applications," Tech. Rep. PLN-4005, Idaho National Laboratory, 2020.

[5] M. DeHart, F. N. Gleicher, V. Labouré, J. Ortensi, Z. Prince, S. Schunert, and Y. Wang, "Mammoth theory manual," Tech. Rep. INL/EXT-19-54252, Idaho National Laboratory, 2017.

[6] Y. Wang, S. Schunert, J. Ortensi, V. Laboure, M. DeHart, Z. Prince, F. Kong, J. Harter, P. Balestra, and F. Gleicher, "Rattlesnake: A moose-based multiphysics multischeme radiation transport application," *Nuclear Technology*, vol. 207, no. 7, pp. 1047–1072, 2021.

[7] Y. Wang, S. Schunert, V. Laboure, and Z. Prince, "Rattlesnake theory manual," Tech. Rep. INL/EXT-17-42103, Idaho National Laboratory, 2017.

[8] C. Lee and W. S. Yang, "MC^2-3: Multigroup cross section generation code for fast reactor analysis," *Nucl Sci Eng*, vol. 187, pp. 268–290, 2017.

[9] Y.S. Jung and C.H. Lee, "PROTEUS-MOC User Manual," Technical Report ANL/NE-18/10, Argonne National Laboratory, September 2018.

[10] C. Lee, Y. S. Jung, Z. Zhong, Y. Wang, J. Ortensi, and M. D. DeHart, "Initial verification and validation database for the NEAMS reactor physics code Griffin," Tech. Rep. ANL/NSE-20/36, INL/LTD-20-59730, Argonne National Laboratory and Idaho National Laboratory, 2020.

[11] Y. Wang, H. Zhang, and R. Martineau, "Diffusion acceleration schemes for the self-adjoint angular flux formulation with a void treatment," *Nuclear Science and Engineering*, vol. 176, pp. 201–225, 2014.

[12] V. M. Laboure, R. G. McClarren, and Y. Wang, "Globally conservative, hybrid self-adjoint angular flux and least-squares method compatible with void.," *Nuclear Science and Engineering*, vol. 185, p. 294–306, February 2017.

[13] C. Lee and Y. S. Jung, "Verification of the cross section library generated using OpenMC and MC^2-3 for PROTEUS," in *PHYSOR 2018: Reactors Physics paving the way towards more efficient systems*, (Cancun, Mexico), Apr. 2018.

[14] I. Diber and E. E. Lewis, "Variational nodal methods for neutron transport," *Nuclear Science and Engineering*, vol. 91, p. 132, 1985.

[15] Y. Wang, S. Schunert, and M. D. DeHart, "Hybrid Pn-Sn calculations with saaf for the multiscale transport capability in rattlesnake," in *Proceeding Physor 2016*, ANS, 2016-05.

[16] S. Schunert, Y. Wang, F. Gleicher, J. Ortensi, B. Baker, V. Laboure, C. Wang, M. DeHart, and R. Martineau, "A flexible nonlinear diffusion acceleration method for the SN transport equations discretized with discontinuous finite elements," *Journal of Computational Physics*, vol. 338, pp. 107–136, jun 2017.

[17] D. Knoll and D. Keyes, "Jacobian-free newton–krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, vol. 193, no. 2, pp. 357–397, 2004.

[18] Y. Wang, J. Ortensi, S. Schunert, and V. Laboure, "A pebble tracking transport algorithm for pebble bed reactor analysis," in *PHYSOR 2018: Reactor Physics paving the way towards more efficient systems*, (Cancun, Mexico), Apr. 2018.

[19] E. E. Lewis, M. A. Smith, and G. Palmiotti, "A new paradigm for local-global coupling in whole-core neutron transport," *Nuclear Science and Engineering*, vol. 161, pp. 279–288, 2009.

[20] M. K. Jaradat, H. Park, W. S. Yang, and C. Lee, "Development and validation of PROTEUS-NODAL transient analyses capabilities for molten salt reactors," *Annals of Nuclear Energy*, vol. 160, p. 108402, 2021.

[21] Y. Wang, "INSTANT theory manual - part I. PN-hybrid-FEM for the multigroup transport equation," tech. rep., Idaho National Laboratory, 2012-02.

[22] E. E. Lewis and G. Palmiotti, "Red-black response matrix acceleration by transformation of interface variables," *Nuclear Science and Engineering*, vol. 130, no. 2, pp. 181–193, 1998.

[23] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc users manual," Tech. Rep. ANL-95/11 - Revision 3.15, Argonne National Laboratory, 2021.

[24] V. Hernandez, J. E. Roman, and V. Vidal, "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Transactions on Mathematical Software*, vol. 31, no. 3, pp. 351–362, 2005-09.

[25] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, "libMesh : a C++ library for parallel adaptive mesh refinement/coarsening simulations," *Engineering with Computers*, vol. 22, no. 3, pp. 237–254, 2006-12.

[26] G. Karypis and V. Kumar, "METIS a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 4.0," tech. rep., University of Minnesota, 1998.

[27] G. Karypis and K. Schloegel, "ParMETIS–Parallel graph partitioning and sparse matrix ordering library, version 4.0," tech. rep., University of Minnesota, 2013. `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`.

[28] X. S. Li and J. W. Demmel, "SuperLU_DIST," *ACM Transactions on Mathematical Software*, vol. 29, pp. 110–140, jun 2003.

[29] P. Amestoy, I. Duff, and J.-Y. L'Excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," *Computer Methods in Applied Mechanics and Engineering*, vol. 184, pp. 501–520, apr 2000.

[30] R. D. Falgout and U. M. Yang, "hypre: A library of high performance preconditioners," in *Computational Science — ICCS 2002* (P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, and J. J. Dongarra, eds.), (Berlin, Heidelberg), pp. 632–641, Springer Berlin Heidelberg, 2002.

[31] T. A. Wareing, J. M. McGhee, J. E. Morel, and S. D. Pautz, "Discontinuous finite element S_N methods on three-dimensional unstructured grids," *Nuclear Science and Engineering*, vol. 138, pp. 256–268, 2001.

[32] Adams, M. L., "Discontinuous finite element transport solutions in thick diffusive problems," *Nuclear Science and Engineering*, vol. 137, no. 3, pp. 298–333, 2001.

[33] Y. Wang, *Adaptive Mesh Refinement Solution Techniques for the Multigroup S_N Transport Equation Using a High-Order Discontinuous Finite Element Method.* PhD thesis, Texas A&M University, 2009.

[34] R. S. Baker and K. R. Koch, "An S_N algorithm for the massively parallel CM-200 computer," *Nuclear Science and Engineering*, vol. 128, pp. 312–320, 1998.

[35] T. S. Bailey and R. D. Falgout, "Analysis of massively parallel discrete-ordinates transport sweep algorithms with collisions (llnl-conf–407968)," in *International Conference on Mathematics, Computational Methods, and Reactor Physics; Saratoga Springs, NY (United States); 3-7 May 2009*, 10 2008.

[36] M. P. Adams, M. L. Adams, W. D. Hawkins, T. Smith, L. Rauchwerger, N. M. Amato, T. S. Bailey, and R. D. Falgout, "Provably optimal parallel transport sweeps on regular grids," in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2013)*, pp. 2535–2553, ANS, 2013.

[37] M. P. Adams, M. L. Adams, W. D. Hawkins, T. Smith, L. Rauchwerger, N. M. Amato, T. S. Bailey, R. D. Falgout, A. Kunen, and P. Brown, "Provably optimal parallel transport sweeps on semi-structured grids," *Journal of Computational Physics*, p. 109234, 2020.

[38] J. I. Vermaak, J. C. Ragusa, M. L. Adams, and J. E. Morel, "Massively parallel transport sweeps on meshes with cyclic dependencies," *Journal of Computational Physics*, vol. 425, p. 109892, 2021.

[39] S. Plimpton, B. Hendrickson, S. Burns, and W. McLendon, "Parallel algorithms for radiation transport on unstructured grids," in *SC '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, pp. 25–25, 2000.

[40] S. J. Plimpton, B. Hendrickson, S. P. Burns, W. M. III, and L. Rauchwerger, "Parallel S_N sweeps on unstructured grids: Algorithms for prioritization, grid partitioning, and cycle detection," *Nuclear Science and Engineering*, vol. 150, pp. 267–283, 2005.

[41] D. Gaston, *Parallel, Asynchronous Ray-Tracing for Scalable, 3D, Full-Core Method of Characteristics Neutron Transport on Unstructured Mesh*. PhD thesis, Massachusetts Institute of Technology, 2020.

[42] Z. Prince, Y. Wang, and L. Harbour, "A diffusion synthetic acceleration approach to k-eigenvalue neutron transport using pjfnk," *Annals of Nuclear Energy*, vol. 148, p. 107714, 2020.

[43] Y. Wang, C. Rabiti, and G. Palmiotti, "Krylov solvers preconditioned with the low-order red-black algorithm for the PN hybrid FEM for the INSTANT code," in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C 2011)*, Latin American Section (LAS) / American Nuclear Society (ANS), May 2011.

[44] P. A. Raviart and J. M. Thomas, "Primal hybrid finite element methods for 2nd order elliptic equations," *MATHEMATICS OF COMPUTATION*, vol. 31, no. 138, pp. 391–413, 1777.

[45] Y. Wang, D. Gaston, A. Lindsay, and V. Laboure, "Optimization of the Rattlesnake code with array variables/kernels," *Transactions of the American Nuclear Society*, vol. 121, pp. 773–776, 2019.

[46] G. A. Freund, H. P. Iskendarian, and D. Okrent, "TREAT, a pulsed graphite-moderated reactor for kinetics experiments," in *Proc. 2nd United Nations Int. Conf. on the Peaceful Uses of Atomic Energy, Geneva, Switzerland*, vol. 10, pp. 461–475, 1958.

[47] J. D. Bess and M. D. DeHart, "Baseline Assessment of TREAT for Modeling and Analysis Needs," Tech. Rep. INL/EXT-15-35372, Idaho National Laboratory, 2015.

[48] C. H. Lee, Y. S. Jung, H. M. Connaway, and T. A. Taiwo, "Simulation of treat cores using high-fidelity neutronics code PROTEUS," in *M&C 2017 - International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Apr. 2017.

[49] J. Leppänen, "Serpent – a continuous-energy Monte Carlo reactor physics burnup calculation code.," tech. rep., VTT Technical Research Centre of Finland, 2015.

[50] M. DeHart, J. Ortensi, and V. Labouré, "NEAMS reactor physics assessment problem," Tech. Rep. INL/LTD-20-59184, Idaho National Laboratory, 2020.