INL/CON-17-40867-Revision-2

A fully coupled two-level Schwarz preconditioner based on smoothed aggregation for the transient multigroup neutron diffusion equations

Fande Kong, Yaqi Wang, Cody J Permann, Sebastian Schunert, John W Peterson, David Andrs, Richard C Martineau

May 2018

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/CON-17-40867-Revision-2

A fully coupled two-level Schwarz preconditioner based on smoothed aggregation for the transient multigroup neutron diffusion equations

Fande Kong, Yaqi Wang, Cody J Permann, Sebastian Schunert, John W Peterson, David Andrs, Richard C Martineau

May 2018

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy

Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Article Type

A fully coupled two-level Schwarz preconditioner based on smoothed aggregation for the transient multigroup neutron diffusion equations

Fande Kong*,[1] Yaqi Wang,[2] Sebastian Schunert,[2] John W. Peterson,[1] Cody J. Permann,[1] David Andrš,[1] and Richard C. Martineau[1]

[1]*Modeling and Simulation, Idaho National Laboratory, P.O. Box 1625, Idaho Falls, ID 83415-3840, USA*
[2]*Nuclear Engineering Methods Development, Idaho National Laboratory, P.O. Box 1625, Idaho Falls, ID 83415, USA*

Correspondence: *Fande Kong, Modeling and Simulation, Idaho National Laboratory, P.O. Box 1625, Idaho Falls, ID 83415, USA. Email: fande.kong@inl.gov; fdkong.jd@gmail.com

Received 26 April 2016; Revised 6 June 2016; Accepted 6 June 2016

Summary

The multigroup neutron diffusion equations (an approximation of the neutron transport equation) are widely used for studying the motion of neutrons and their interactions with stationary background materials. Solving the multigroup neutron diffusion equations is challenging because the unknowns are tightly coupled through scattering and fission events, and solutions with high spatial resolution of full reactor cores in multiphysics environments are frequently required. In this paper, we focus on the development of a scalable, parallel preconditioner for solving the system of equations arising from the finite element discretization of the multigroup neutron diffusion equations in space and an implicit finite difference scheme in time. The parallel preconditioner (here referred to as the "fully coupled Schwarz preconditioner") is constructed by monolithically applying the overlapping domain decomposition method together with a smoothed aggregation-based coarse space to the coupled system. Our approach is different from the traditional block Gauss-Seidel sweep method that applies the preconditioner from the fast group to the thermal group sequentially, and we demonstrate that it provides significant improvements in terms of both the

number of iterations required and the total compute time for a system of equations with millions of unknowns on a large supercomputer.

Keywords: parallel processing, two-level Schwarz preconditioner, multigroup neutron diffusion equations, Newton-Krylov-Schwarz, smoothed aggregation, coarse space, finite element method

1 Introduction

The accurate prediction of the (angular or scalar) neutron flux is essential to the design of nuclear reactors and for their safe and economic operation [8, 19]. The angular neutron flux is a quantity corresponding to the product of the neutron speed and the neutron density, and is defined in a seven-dimensional phase space (3D space, 1D time, 2D direction of motion, and 1D energy). The linear Boltzmann equation or the radiation transport equation is used to describe the angular neutron flux distribution. Because the neutron energy spans ten orders of magnitude ranging from 0.001 eV to 20 MeV in a typical nuclear reactor, the computing resources required for performing multi-dimensional transport calculations with continuous or point-wise energy resolution is prohibitive for any real application [8].

Instead, the multigroup approximation is typically applied, where continuous-energy cross sections, used to characterize the probability per unit path length for a nuclear reaction to occur, are collapsed for a selected set of energy ranges (also referred to as energy groups). The multigroup cross sections are produced by integrating the energy dependent cross sections and the neutron energy spectrum over the extent of an energy group. The spectra are typically obtained from a separate lattice calculation [8].

In the present work, the multigroup cross sections are taken to be given parameters. The multigroup approximation to the transport equation (referred to as "multigroup neutron transport equations") is still prohibitively expensive for a full-core nuclear reactor simulation, and is further simplified to the multigroup diffusion equations by introducing a scalar flux variable, which is defined as the integral of the angular neutron fluxes over all the directions of motion. The multigroup neutron diffusion equations are well suited to modeling nuclear reactor cores with significant spatial homogenization, and are much less expensive to solve than the neutron transport equation [8, 19]. However, as we stated earlier, it is still challenging to solve the multigroup neutron diffusion equations in certain applications.

Fast, computationally efficient solvers of the multigroup neutron diffusion equations require scalable, parallel algorithms which take advantage of the capabilities of modern supercomputers. In this paper, we propose a solver consisting of an inexact Jacobian-free Newton method [14] for the system of nonlinear equations, and a Krylov subspace method [23] for the solution of the Jacobian system, using a Schwarz preconditioner to improve convergence.

Over the past few decades, there have been many research articles on the various computational approaches to numerical simulation of the neutron diffusion equations. Both finite element [27, 32], and finite difference [2] methods

have been used successfully for the spatial discretization, typically in conjunction with the power iteration for computing the eigenvalue. A comparison between high order finite element methods and the finite difference method is described in [12]. Parallel algorithms based on the preconditioned BiCGStab solver were developed in [26] for the transient multigroup neutron diffusion equations based on a finite volume discretization in space, and a combined Crank-Nicholson/BDF2 discretization in time, and the resulting algorithm was shown to scale to 12 processors. Regardless of the discretization method, the most computationally expensive part of the simulation is solving large, often ill-conditioned, linear systems repeatedly. We propose a fully coupled overlapping domain decomposition method for the large linear system in this paper.

Domain decomposition methods have been receiving increased attention in the nuclear engineering community due to their ability to solve large systems of equations in parallel. In [5] a non-overlapping domain decomposition method based on Lagrange multipliers is applied to the simplified transport equations. In [11], a non-overlapping Schwarz method is studied for the one-speed neutron diffusion equation, and the Robin interface condition is used to exchange data across subdomains. A hierarchical domain decomposition together with the boundary element method is employed for the neutron diffusion equations on the multiregions in [21]. In [31], a Schwarz and substructuring based Schur complement method is studied for the neutron diffusion equations, and the domain decomposition based preconditioners are shown to work better than those based on an incomplete LU factorization.

Most published works apply the domain decomposition method to the single-group diffusion equation obtained by decoupling the multigroup diffusion equations. For example, in [5, 31] an inverse power iteration is employed to solve the k-eigenvalue problems, and during each power iteration, a Gauss-Seidel iteration is used to sweep through groups from the fast group to the thermal group. In this case, the domain decomposition based preconditioner is applied to each group separately. This decoupled application of the preconditioner can degrade the overall efficiency of the algorithm in certain applications.

In this paper, we propose a fully coupled Schwarz preconditioner for the multigroup neutron diffusion equations. To further improve the parallel performance for large-scale problems, a coarse space based on smoothed aggregation is introduced to construct a two-level method. Compared with the one-level method, the two-level version performs better in terms of the compute time and the number of linear iterations required. Note that the eigenvalue and the transient problems are solved using the same algorithm framework, namely, Newton-Krylov-Schwarz. The fully coupled Schwarz serves as the preconditioner of the linear solvers for both the eigenvalue and the transient problems. We also note that the fully coupled Schwarz preconditioner has been successfully applied to elasticity [16] and fluid-structure interaction [17, 18] problems. Here, we extend and adapt the algorithm to the transient multigroup neutron diffusion equations.

The remainder of this paper is organized as follows. In Section 2, we present the multigroup neutron diffusion equations and their spatial and temporal discretizations. In Section 3, we discuss a parallel algorithm for solving the discretized nonlinear equations (and associated nonlinear eigenvalue system), and the corresponding preconditioned Krylov subspace method. We report on the parallel performance of the proposed algorithm in Section 4, and finally conclusions are drawn in Section 5.

2 Multigroup neutron diffusion equations

As mentioned previously, the multigroup neutron diffusion equations are used to study the motion of particles and their interactions with stationary background materials[8, 19]. Denoting the scalar neutron flux of energy group g by Φ_g [$\mathrm{cm}^{-2}\mathrm{s}^{-1}$], the multigroup neutron diffusion equations are

$$\left.\begin{aligned}
&\frac{\partial}{\partial t}\left(\frac{\Phi_g}{v_g}\right) - \nabla\cdot D_g\nabla\Phi_g + \Sigma_{r,g}\Phi_g = \\
&\qquad Q_{g,0} + \sum_{\substack{g'=1\\ g'\neq g}}^{G}\Sigma_{s,g'\to g}\Phi_{g'} + (1-\beta)\chi_{p,g}\sum_{g'=1}^{G}\nu\Sigma_{f,g'}\Phi_{g'} + \sum_{i=1}^{I}\chi_{d,i,g}\lambda_i C_i \\
&\frac{\partial C_i}{\partial t} = \beta_i\sum_{g'=1}^{G}\nu\Sigma_{f,g'}\Phi_{g'} - \lambda_i C_i \\
&\Phi_g = \Phi_g^0 \qquad \text{at } t=0 \\
&C_i = \frac{\beta_i\sum_{g'=1}^{G}\nu\Sigma_{f,g'}\Phi_{g'}^0}{\lambda_i} \qquad \text{at } t=0 \\
&\Phi_g = \Phi_{g,d} \quad \text{in } \Gamma_d \\
-D_g\Phi_g\cdot\boldsymbol{n} &= J_{g,n} \quad \text{in } \Gamma_n = \partial\Omega\setminus\Gamma_d,
\end{aligned}\right\} \tag{1}$$

where, unless otherwise noted, the equations apply on the entire spatial domain Ω, I is the number of delayed neutron precursor groups (6 in this paper), G is the number of energy groups (11 in this paper), v_g [$\mathrm{cm\,s}^{-1}$] is the group averaged neutron speed (that is, the neutron speed averaged for all neutrons over the group g, indicating how fast on average the neutrons of the gth group move), D_g [cm] is the diffusion coefficient, $\Sigma_{r,g}$ [cm^{-1}] is the macroscopic removal cross section, $Q_{g,0}$ [$\mathrm{cm}^{-3}\mathrm{s}^{-1}$] is the external source, $\Sigma_{s,g'\to g}$ [cm^{-1}] denotes the macroscopic scattering cross section from group g' to group g, β_i is the delayed neutron fraction, $\beta = \sum_{i=1}^{I}\beta_i$ is the total delayed neutron fraction, $\chi_{p,g}$ is the prompt fission spectrum, ν is the average number of neutrons emitted per fission, $\Sigma_{f,g}$ [cm^{-1}] is the macroscopic fission cross section, λ_i [s^{-1}] is the decay constant, $\chi_{d,i,g}$ is the delayed fission spectrum, and C_i [cm^{-3}] is the concentration of delayed neutron precursors. $\Phi_{g,d}$ [$\mathrm{cm}^{-2}\mathrm{s}^{-1}$] is the given scalar flux on Dirichlet boundary Γ_d, and $J_{g,n}$ [$\mathrm{cm}^{-2}\mathrm{s}^{-1}$] is the net current on Neumann boundary Γ_n.

The set of equations (1) represents the neutron balance for all energy groups. In the first equation of (1), the first term on the left hand side represents the rate of change of neutron population in the group g, the second term is the leakage rate due to diffusion (referred to as the "diffusion term") and the third is the neutron removal by collision ("removal term"). The second term on the right hand side represents the neutrons transferred via scattering into group g from other groups ("scattering term"); this term couples the fluxes of all groups together. The third term of the right hand side is the fission neutron production rate, and the fourth term represents the number of the neutrons from the delayed neutron precursors (DNP). Most neutrons born from fission reactions appear instantaneously, but a fraction appear delayed as the decay products of fission product nuclides, and are referred to as neutron precursors.

To simulate the transient behavior of nuclear reactors, it is essential to track neutron precursor concentrations. Neutron precursors are separated into groups by their decay constants, and these constants govern their dynamic behavior. The fission, scattering, and removal cross sections and the diffusion coefficient D_g depend, in a complicated way, on the temperature within a multiphysics environment. For real applications, the explicit form of this dependence is not available, and we resort to interpolating the cross section values using pre-generated tabulated data.

More precisely, in this work, a few pairs $(\Sigma_{\cdots}^{(j)}, T^{(j)})$ are provided during multigroup cross section generations, and for a given temperature T, the cross sections are calculated using a linear interpolation with $\Sigma_{\cdots}^{(j)}$ and $\Sigma_{\cdots}^{(j+1)}$ if $T^{(j)} < T < T^{(j+1)}$. The temperature T is obtained by solving the following differential equation:

$$\rho c_p \frac{\partial T}{\partial t} = \sum_{g'}^{G} \kappa \Sigma_{f,g'} \Phi_{g'}, \tag{2}$$

where c_p [$\mathrm{Jg^{-1}K^{-1}}$] is the heat capacity, which, for the nuclear fuel, depends on the temperature according to

$$c_p = -5.8219 \times 10^{-10} T^3 - 4.3694 \times 10^{-7} T^2$$

$$+ 2.8369 \times 10^{-3} T - 1.009 \times 10^{-2}. \tag{3}$$

The material density is ρ [$\mathrm{gcm^{-3}}$], and κ [J] is the energy released per fission. The right hand side of (2) is also referred to as the "power density", ρ_{power} [$\mathrm{Wcm^{-3}}$]. In Equation (2), we assume that the transient is so fast (on the order of a few seconds) that the heat generated from fission does not have time to diffuse away.

In Equation (1), Φ_g^0 is the initial condition that is obtained by solving the generalized eigenvalue problem:

$$-\nabla \cdot D_g \nabla \Phi_g^0 + \Sigma_{r,g} \Phi_g^0 = \sum_{g' \neq g} \Sigma_{s,g' \to g} \Phi_{g'}^0 + \frac{1}{k} \chi_g \sum_{g'} \nu \Sigma_{f,g'}^0 \Phi_{g'}^0, \tag{4}$$

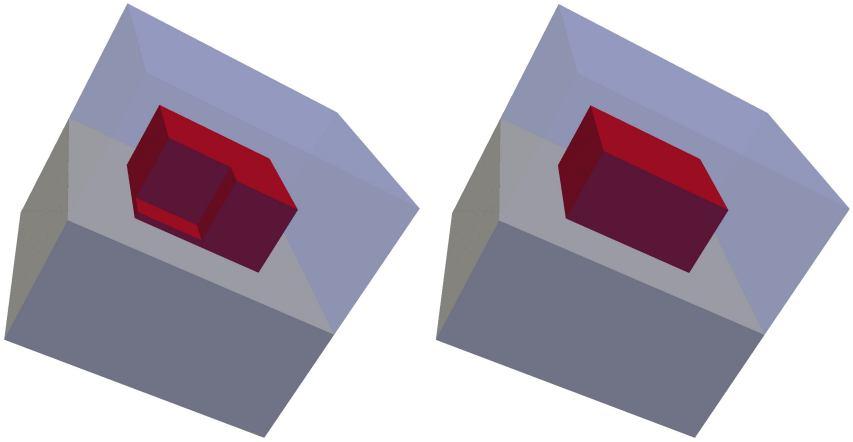

Figure 1: Computational domain for the eigenvalue problem and the transient problem. Left: computational domain for the generalized eigenvalue problem, right: computational domain for the transient problem. The red cube is the nuclear fuel; the gray part is graphite.

where $\chi_g = (1-\beta)\chi_{p,g} + \sum_{i=1}^{I} \beta_i \chi_{d,i,g}$ is the average spectrum. Here k is typically used to adjust the fission cross sections, $\Sigma_{f,g} = 1/k\Sigma^0_{f,g}$, for the transient Equation (1) so that the loss and production of neutrons in the system is balanced.

We call the maximum k (corresponding to the smallest eigenvalue of (4)) and its corresponding eigenvector the "fundamental mode" of the generalized eigenvalue problem. k is also referred to as the system's multiplication factor. Equation (4) is defined on the same computational domain as Equation (1), and we ignore the temperature dependence by assuming the power is low. The transient is initiated by substituting a block of graphite with fuel as indicated in Fig. 1. This is used to mimic the generation of a transient by pulling out a control-rod in the nuclear reactor.

To discretize (1) and (4) in space, a hexahedral mesh Ω_h is generated for the computational domain Ω, and the standard Galerkin finite element method is employed. The corresponding semi-discrete systems of equations for (1) are written as

$$\frac{d\boldsymbol{y}(t)}{dt} = N_{yy}(\boldsymbol{y}(t)) + N_{yc}(\boldsymbol{c}(t)) + F \tag{5}$$

$$\frac{d\boldsymbol{c}(t)}{dt} = N_{cy}(\boldsymbol{y}(t)) + N_{cc}(\boldsymbol{c}(t)) \tag{6}$$

where $\boldsymbol{y}(t) = \{y_1, y_2, ..., y_G\}$ is a vector of nodal values of $\{\Phi_1, \Phi_2, ..., \Phi_G\}$ at time t, and $N_{yy}(\boldsymbol{y}(t))$ corresponds to all the terms of the first equation of (1) except the time derivative, the DNP terms and the external source term. F is the external source term, and $N_{yc}(\boldsymbol{c}(t))$ is the DNP term coupling the variables $\boldsymbol{c}(t)$ and $\boldsymbol{y}(t)$. $\boldsymbol{c}(t) = \{c_1, c_2, ..., c_I\}$ is the discrete version of $\{C_1, C_2, .., C_I\}$ that is a function of the scalar fluxes, $\boldsymbol{y}(t)$, satisfying the second equation

of (1). N_{cy} couples $\boldsymbol{y}(t)$ and $\boldsymbol{c}(t)$, and corresponds to the first term on the right hand side of the second equation of (1), while $N_{cc}(\boldsymbol{c}(t))$ corresponds to the second term on the right hand side.

The Crank-Nicolson method is used to discretize (5) in time, resulting in

$$M\boldsymbol{y}_{n+1} - \frac{\delta t}{2}(N_{yy}(\boldsymbol{y}_{n+1}) + N_{yc}(\boldsymbol{c}_{n+1}) + F_{n+1}) = M\boldsymbol{y}_n + \frac{\delta t}{2}(N_{yy}(\boldsymbol{y}_n) + N_{yc}(\boldsymbol{c}_n) + F_n), \tag{7}$$

where y_{n+1} is the solution at time step $n+1$, and δt is the time step size. F_{n+1} is the external source at the $(n+1)$st step and M is the mass matrix. In order to compute $\boldsymbol{c}_{n+1}$, the second equation of (1) is dicretized by backward Euler in time,

$$\boldsymbol{c}_{n+1} = \delta t(N_{cy}(\boldsymbol{y}_{n+1}) + N_{cc}(\boldsymbol{c}_{n+1})) + \boldsymbol{c}_n, \tag{8}$$

where $\boldsymbol{c}_{n+1}$ is the solution at the $(n+1)$st time step. With a given initial concentration as in (1), $\boldsymbol{c}_{n+1}$ is computed using (8) for a given $\boldsymbol{y}_{n+1}$ by inverting the diagonal matrix N_{cc}. To save memory while solving the coupled system defined by (7) and (8), the variable $\boldsymbol{c}_{n+1}$ is eliminated from (7) by rewriting (8) as

$$\boldsymbol{c}_{n+1} = (\boldsymbol{I} - \delta t N_{cc})^{-1}(\delta t N_{cy}(\boldsymbol{y}_{n+1}) + \boldsymbol{c}_n). \tag{9}$$

Since $(\boldsymbol{I} - \delta t N_{cc})$ is a diagonal matrix, the computation of the inverse matrix is straightforward and inexpensive. Therefore, we substitute (9) into (7), and the resulting system is a nonlinear system of equations with scalar fluxes, $\boldsymbol{y}_{n+1}$, as its independent variables.

We also do a similar elimination for the temperature equation. The temperature equation is different from the delayed neutron precursor equation in two respects. First, Equation (2) is a nonlinear differential equation because both c_p (heat capacity) and the fission cross sections $\Sigma_{f,g}$ depend on the temperature T. Second, the updated temperature will be used to evaluate the cross sections, thereby making the system (1) nonlinear. Using the Trapezoidal rule, equation (2) is discretized as:

$$\rho c_p(T^{n+1})(T^{n+1} - T^n) = \frac{\delta t}{2} N_{Ty}(\boldsymbol{y}_{n+1} + \boldsymbol{y}_n) \tag{10}$$

where N_{Ty} corresponds to the right hand side of (2). The system of nonlinear equations (10) is solved by a local Newton iteration at each quadrature point. No communication is involved in this local Newton solve because only local data is required, and since there is only one unknown per Newton solve, no linear solver is required. We refer

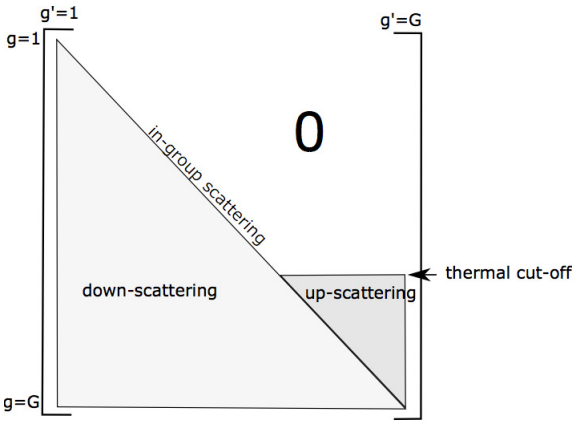

Figure 2: Pattern of neutron scattering matrix.

to this Newton iteration as "local" to distinguish it from the global Newton iteration (to be introduced in the next section) that is used for solving the nonlinear equation (7).

For simplicity, we rewrite (7) as

$$\mathcal{F}(\boldsymbol{y}) = 0, \tag{11}$$

ignoring the $n+1$ subscript. Equation (11) is solved by a Jacobian-free Newton method as in [15]. We note that (11) is nonlinear because of the temperature feedback (10). If we do not consider the temperature feedback, Equation (11) is linear. The parallel algorithm we develop here is general enough to be applicable for both nonlinear and linear problems.

The corresponding Jacobian system for the nonlinear system (11) is difficult to compute because

1. The group variables are coupled through scattering, as shown in Fig. 2, and fission events. The fission cross sections in the nuclear fuel are non-zero for all energy groups and therefore couple together all the group variables (although fission neutrons are mainly present in the fast energy groups).

2. The material coefficients (including the cross sections and the diffusion coefficients) in Equation (1), shown in Tables 8–13 in the Appendix, are discontinuous across the graphite/nuclear fuel interface.

3. If temperature feedback is considered, then the fluxes depend on the temperature and vice-versa.

To solve the nonlinear system (11), an accurate representation of the Jacobian matrix is required to compute the Newton descent direction, otherwise the solver will require a suboptimal number of Newton iterations, or possibly diverge. To overcome this difficulty, we employ the Jacobian-free version of Newton's method that approximates the action of the Jacobian using finite differences. The method is described in detail in the next section.

3 Scalable parallel algorithm framework

There are two systems of equations to be solved: the generalized eigenvalue problem (12) and the system of nonlinear equations (11). We first present a parallel eigenvalue solver for computing the fundamental mode. The eigenvalue solver is further accelerated by converting the eigenvalue system to a system of nonlinear equations. Finally, we describe a fully coupled nonlinear solver together with a parallel preconditioner.

3.1 Eigenvalue solver

The generalized eigenvalue problem (4) takes the following form after spatial discretization:

$$\mathcal{A}\boldsymbol{y}_0 = \frac{1}{k}\mathcal{B}\boldsymbol{y}_0, \tag{12}$$

where $\mathcal{B}$ corresponds to the fission in (4), and $\mathcal{A}$ represents the other terms. We remark that (12) is a linear eigenvalue problem in the present work, but it is a system of nonlinear equations if we insert $k = \|\mathcal{B}\boldsymbol{y}_0\|$ into (12), and therefore our solver is designed with both scenarios in mind.

The simplest algorithm for computing the smallest eigenvalue is the inverse power iteration [24]. The inverse power iteration works well for problems in which the ratio between the smallest and second smallest eigenvalues is much smaller than 1, otherwise it may converge very slowly. An improved algorithm based on Newton's method is used in such a case [15]. More precisely, the eigenvalue problem is reformulated as a nonlinear problem

$$\mathcal{A}\boldsymbol{y}_0 = \frac{1}{\|\mathcal{B}\boldsymbol{y}_0\|}\mathcal{B}\boldsymbol{y}_0. \tag{13}$$

where $k \equiv \|\mathcal{B}\boldsymbol{y}_0\|$. An inexact Newton method (to be described in next section) is employed to solve Equation (13). Newton's method converges quadratically if the initial guess is sufficiently close to the solution. An effective way to compute an initial guess for (13) is to apply a few inverse power iterations.

3.2 Newton-Krylov-Schwarz

We next describe a parallel algorithm framework for solving (11) and (13), followed by a coarse space to improve convergence and scalability. A Jacobian-free Newton method [14] is used for solving the nonlinear system (11) and (13). During each Newton iteration, the Jacobian system is computed using a Krylov subspace method [23], e.g. GMRES [25], together with a Schwarz preconditioner to be described shortly. More precisely, the solution at the current Newton step, $\boldsymbol{y}^{(k+1)}$, is updated by adding a Newton descent direction $\delta\boldsymbol{y}^{(k)}$ to the previous Newton step solution $\boldsymbol{y}^{(k)}$, that

is,

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} + \alpha^{(k)} \delta \boldsymbol{y}^{(k)}, \tag{14}$$

where $\alpha^{(k)}$ is a linesearch step size calculated using a backtracking method [7], and $\delta \boldsymbol{y}^{(k)}$ is obtained by approximately solving the Jacobian system,

$$\mathcal{J}(\boldsymbol{y}^{(k)}) \delta \boldsymbol{y}^{(k)} = -\mathcal{F}(\boldsymbol{y}^{(k)}), \tag{15}$$

where $\mathcal{F}(\boldsymbol{y}^{(k)})$ is the nonlinear function residual evaluated at $\boldsymbol{y}^{(k)}$, and $\mathcal{J}(\boldsymbol{y}^{(k)})$ is the Jacobian matrix evaluated at $\boldsymbol{y}^{(k)}$, although it is not explicitly formed. The action of $\mathcal{J}(\boldsymbol{y}^{(k)})$ on a vector $\boldsymbol{r}$ is computed via finite differences as

$$\mathcal{J}(\boldsymbol{y}^{(k)}) \boldsymbol{r} \approx \frac{\mathcal{F}(\boldsymbol{y}^{(k)} + \gamma \boldsymbol{r}) - \mathcal{F}(\boldsymbol{y}^{(k)})}{\gamma}, \tag{16}$$

where γ is a small parameter. To speed up the convergence of the Krylov subspace method used in solving (15), we construct a Schwarz preconditioner based on the matrix $B_h \approx \mathcal{J}(\boldsymbol{y}^{(k)})$. We neglect the derivatives of the material coefficients with respect to the fluxes when computing B_h, because their analytical forms are not available. As stated earlier, the material coefficients depend on the temperature through interpolation of the tabulated data, and the temperature is a nonlinear function of the fluxes. Neglecting the derivatives greatly simplifies the computation of B_h while still retaining the operator's effectiveness as a preconditioner.

The right-preconditioned Jacobian system is defined as

$$\mathcal{J} B_h^{-1} B_h \delta \boldsymbol{y} = -\mathcal{F} \tag{17}$$

where we ignore the superscript k and arguments for $\mathcal{J}$, $\delta \boldsymbol{y}$ and $\mathcal{F}$ to simplify the notation. The preconditioning procedure is accomplished via two substeps. First, $\mathcal{J} B_h^{-1} \boldsymbol{r} = -\mathcal{F}$ is solved for $\boldsymbol{r}$ using GMRES together with one preconditioner application per GMRES iteration, and then $B_h \delta \boldsymbol{y} = \boldsymbol{r}$ is solved for $\delta \boldsymbol{y}$ via one application of the preconditioner. We will next discuss the construction of the preconditioner.

We consider two approaches to constructing the preconditioner for the multigroup neutron diffusion equations. The first approach, the block Gauss-Seidel algorithm, involves solving Equation (17) in a group-by-group manner. Let us consider one application of the preconditioner for the equation $B_h \delta \boldsymbol{y} = \boldsymbol{r}$. In group-by-group notation, this

equation is given by:

$$\begin{bmatrix} B_{h,11} & B_{h,12} & B_{h,13} & ... & B_{h,1G} \\ B_{h,21} & B_{h,22} & B_{h,23} & ... & B_{h,2G} \\ B_{h,31} & B_{h,32} & B_{h,33} & ... & B_{h,3G} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ B_{h,G1} & B_{h,G2} & B_{h,G3} & ... & B_{h,GG} \end{bmatrix} \begin{bmatrix} \delta y_1 \\ \delta y_2 \\ \delta y_3 \\ \vdots \\ \delta y_G \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_G \end{bmatrix}, \tag{18}$$

where $B_{h,gg'}$ represents a submatrix of B_h coupling groups g and g', δy_g is a subvector of $\delta \boldsymbol{y}$ for the group g, and r_g is a subvector of $\boldsymbol{r}$ corresponding to the group g. The block Gauss-Seidel algorithm is carried out with $\delta \boldsymbol{y} = 0$, and solve the first group equation for δy_1, then substitute it into the right hand side of the second group equation, solve for δy_2, etc. for the remaining groups.

The block Gauss-Seidel algorithm was traditionally preferred because the individual subproblems are small and can be solved on limited memory computers. In contrast, on large supercomputers, in our numerical results it appears better to solve for all of the group fluxes simultaneously by employing a scalable, parallel preconditioner. The preconditioner must be designed in such a way that the computational load remains well-balanced and the cost and amount of inter-processor communication is minimized.

In the present work, we construct the preconditioner B_h^{-1} based on a fully coupled overlapping domain decomposition method. The basic idea of the overlapping domain decomposition method[6, 22, 28, 29] is to partition the computational domain (mesh) Ω_h into n_p subdomains $\Omega_{h,i}$, and then each subdomain is extended to overlap with its neighbors by a number of layers we denote by δ. This extension is accomplished without using information from the mesh. Instead, a graph derived from the sparsity pattern of the preconditioning matrix B_h is employed.

We now briefly describe the fully coupled DDM. We define the block index set $S_h = \{S_h^1, S_h^2, S_h^3, ..., S_h^{n_{\text{node}}}\}$ corresponding to the global degree of freedom indices, where the S_h^i represent sets of variable indices associated with mesh node i, and n_{node} is the number of total mesh nodes. S_h is distributed across n_p processors by dividing it into subsets $S_i, i = 1, 2, ..., n_p$ such that $S_h = \cup_{i=1}^{n_p} S_i$, $S_i \cap S_j = \emptyset$ when $i \neq j$.

The block structure of S_h is taken into account when S_h is distributed so that each block S_h^i is owned by a single processor. It is advantageous to preserve the tightly coupled block structure during partitioning since the physics are naturally coupled together in the same manner. S_i is extended to overlap with its neighbors by δ layers, where each block is treated as a single unknown, and the corresponding overlapping subset is denoted as S_i^δ. The same idea is applied to the construction of the subdomain vector $\boldsymbol{r}_{h,i}^\delta$ and the restriction operator $R_{h,i}^\delta$. Here, the operator $R_{h,i}^\delta$ is used to extract the corresponding components for $\boldsymbol{r}_{h,i}^\delta$ from a global vector $\boldsymbol{r}_h$. The submatrix $B_{h,i}^\delta$ is extracted

from its global counterpart using $R^{\delta}_{h,i}$ as well, that is,

$$B^{\delta}_{h,i} = R^{\delta}_{h,i} B_h (R^{\delta}_{h,i})^T, \quad i = 1, 2, ..., n_p.$$

Based on these components, a fully coupled restricted additive Schwarz (RAS) preconditioner is given by

$$B^{-1}_{\text{one}} = \sum_{i=1}^{n_p} (R^0_{h,i})^T (B^{\delta}_{h,i})^{-1} R^{\delta}_{h,i}, \tag{19}$$

where $R^0_{h,i}$ returns the subvectors defined on the non-overlapping subdomains, and $(B^{\delta}_{h,i})^{-1}$ represents a subdomain solver, which is an incomplete LU (ILU) factorization in this work. The ILU factorization can employ a specific submatrix reordering scheme (e.g. nested dissection (ND), one-way dissection (1WD), quotient minimum degree (QMD), and reverse Cuthill-McKee (RCM)[23]). Sometimes the reordering schemes improve the performance of the algorithm. $(R^0_{h,i})^T$ discards the overlapping part of the solution to reduce communication and improve the convergence of the algorithm.

The performance of the one-level method can be further improved by introducing a coarse space to construct a two-level method. Let us denote the coarse space as S_H, and a prolongation operator from the coarse space to the fine space as P^h_H. The transpose of the prolongation operator, $(P^h_H)^T$, is used as the restriction operator from the fine space to the coarse space. B_H represents a matrix defined on the coarse space S_H. We denote the two-level Schwarz preconditioner as B^{-1}_{two}, and its action on a vector is implemented via Algorithm 1, that is,

$$\delta \boldsymbol{y}_h = B^{-1}_{\text{two}} \boldsymbol{r}_h. \tag{20}$$

At line 4 of Algorithm 1, B^{-1}_H represents a coarse solver that is another restricted Schwarz with ILU as the subdomain solver in this work. B^{-1}_h is B^{-1}_{one} when using the one-level method, and it is B^{-1}_{two} for the two-level method.

Algorithm 1 Two-level Schwarz preconditioner B^{-1}_{two}.

1: Input a residual from the outer solver $\boldsymbol{r}_h$
2: Solve $\delta \boldsymbol{y}^{(1/3)}_h = B^{-1}_{\text{one}} \boldsymbol{r}_h$
3: Compute $\boldsymbol{r}_H = (P^h_H)^T (\boldsymbol{r}_h - B_h \delta \boldsymbol{y}^{(1/3)}_h)$
4: Solve $\delta \boldsymbol{y}^{(2/3)}_h = \delta \boldsymbol{y}^{(1/3)}_h + P^h_H B^{-1}_H \boldsymbol{r}_H$
5: Compute $\bar{\boldsymbol{r}}_h = \boldsymbol{r}_h - B_h \delta \boldsymbol{y}^{(2/3)}_h$
6: Solve $\delta \boldsymbol{y}_h = \delta \boldsymbol{y}^{(2/3)}_h + B^{-1}_{\text{one}} \bar{\boldsymbol{r}}_h$
7: Return $\delta \boldsymbol{y}_h$

We mention that a parallel RAS with ILU as the subdomain solver is chosen as the block solver of the block Gauss-Seidel method. In Guass-Seidel, the parallel RAS is applied group-by-group, and it is different from its fully coupled version where all the group fluxes are computed simultaneously.

3.3 Smoothed aggregation based Schwarz coarse space

The coarse space S_H plays a critical role, and the algorithm performs poorly if a bad coarse space is used. We use the smoothed aggregation (SA) method, as discussed in [1, 30], to construct the coarse operator B_H for Algorithm 1. The basic idea of SA is to decompose the entire graph into a number of disjoint aggregates, where each aggregate corresponds to an unknown on the coarse space. The coarse operator B_H is computed using the Galerkin method,

$$B_H = (P_H^h)^T B_h P_H^h.$$

More precisely, a graph, denoted as $\mathcal{G} = \{\mathrm{v}_i, e_{ij}\}$, is constructed based on the numerical values of B_h such that an edge e_{ij} between the unknowns v_i and v_j is formed if $|(B_h)_{ij}| > \omega\sqrt{|(B_h)_{ii}||(B_h)_{jj}|}$. Here $\omega \in [0, 1)$ is a parameter which controls which values are dropped; smaller values of ω correspond to more connections being kept. We apply a maximum independent set (MIS) algorithm to partition $\mathcal{G}$ into a number of aggregates, $\{V_i\}$, such that $V_i \cap V_j = \emptyset$, $i \neq j$, and $\cup V_i = \mathcal{M}$, where $\mathcal{M}$ is the set of all nodes in $\mathcal{G}$. The aggregate, V_i, collapses to form a coarse space node i. The dimension of $\{V_i\}$ might not be small enough after applying MIS, so a post-processing procedure, as suggested in [1], may be carried out. This procedure tries to merge small aggregates together if they are strongly connected, in order to reduce the dimension of the coarse space. An initial prolongation operator $\tilde{P}_H^h$ is formed using the aggregates as follows

$$(\tilde{P}_H^h)_{ij} = \begin{cases} 1 & \text{if } i \in V_j, \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

The final prolongation P_H^h is obtained by improving $\tilde{P}_H^h$ with a Jacobi smoother, that is,

$$P_H^h = (I - \eta(\mathrm{diag}(B_h))^{-1} B_h)\tilde{P}_H^h, \tag{22}$$

where $\mathrm{diag}(B_h)$ represents the diagonal part of B_h, and η is the smoothing parameter, which is chosen as $1.5\theta^{-1}$, where θ is an estimate of the largest eigenvalue of $\mathrm{diag}(B_h)^{-1}B_h$ (interested readers are referred to [1] for more details). In the next section we will demonstrate, via numerical experiments, that the two-level preconditioner based on the coarse space and built using the smoothed aggregation approach is more efficient than the one-level method.

4 Numerical experiments

In this section, we discuss the performance of the proposed algorithm by simulating a graphite moderated nuclear reactor. The simulation is carried out on a supercomputer consisting of Intel Xeon E5-2680 v3 CPUs connected by a low latency InfiniBand network. In the following discussion, the term "NI" refers to the average number of Newton iterations per time step, "LI" is the average number of GMRES iterations per Newton step, "Time" denotes the total compute time in seconds, "EFF" is the parallel efficiency with respect to the number of processors, and "n_p" is the number of processors.

ILU(0) and $\delta = 1$ are used as the default subdomain solver configuration, unless otherwise specified. The relative stopping condition for the Newton iterations is 10^{-8} and that for the linear solver is 10^{-3}. The spatial and temporal discretizations are based on Rattlesnake[33], MOOSE[10] and libMesh[13], and the nonlinear and linear solvers are implemented on top of PETSc[4]. The simulation is carried out with $\delta t = 10^{-2}$ s for $t \in (0, 0.1]$ for the reactor core as shown in Figs. 3 and 4.

The test problem we consider is a simplified version of the TREAT (the Transient Test Reactor at the Idaho National Laboratory) calibration transient 15[20], which consists of a cubic fuel core surrounded by graphite reflectors which are 200 and 400 cm across, respectively. Reactivity is increased by adding more fuel to the reactor and removing some graphite such that the reactor becomes super-critical and starts the transient. The fuel consists of highly-enriched Uranium particles in a graphite matrix.

TREAT is a "thermal" reactor, which means that the bulk of the fissions are caused by thermal neutrons. The particular challenge when simulating TREAT is the upscattering of neutrons in the graphite. In upscattering, neutrons are scattered from lower to higher energies. The Gauss-Seidel sweep moves from faster energies to slower energies and therefore resolves downscattering, but lags upscattering. The fully coupled approach resolves both downscattering and upscattering simultaneously.

The initial conditions, generated by solving an eigenvalue problem, are shown in Fig. 3, and the solutions at $t = 0.04$ s and 0.06 s for the transient problem are given in Fig. 4. We treat the initial solve of the eigenvalue problem as an extra time step when reporting the number of Newton iterations, GMRES iterations, and compute time. The cross section and fission rate parameters for (1) and (4) are given in[3], and in the Appendix.

4.1 Comparison with Gauss-Seidel sweeps

The Gauss-Seidel sweep algorithm has traditionally been employed as an inner solver within the power iteration, but it is not suitable for parallel processing since it sweeps through the various groups sequentially. In this section, we compare the performance of the fully coupled Schwarz preconditioner with that of the Gauss-Seidel sweep. A mesh

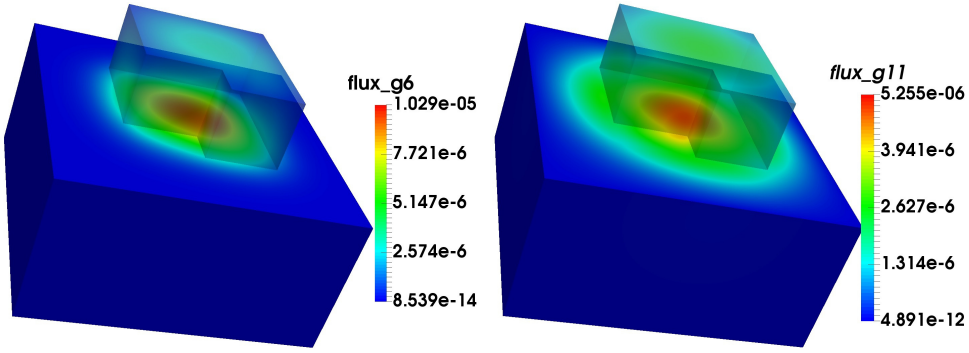

Figure 3: Scalar flux for the 6th (left) and 11th (right) groups for the eigenvalue problem.

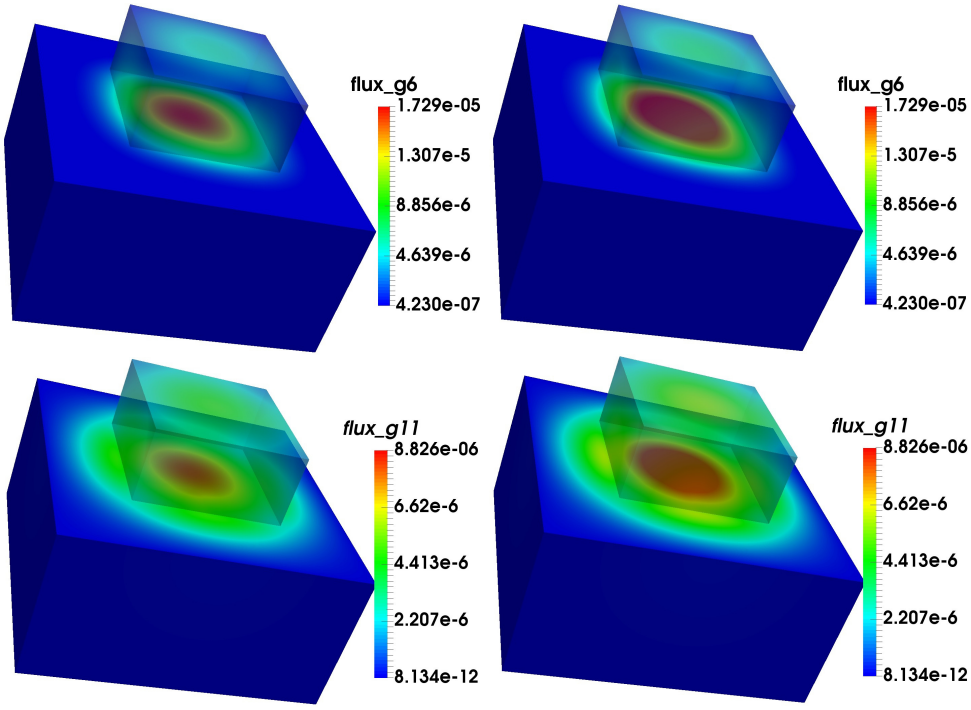

Figure 4: Scalar flux of the 6th (top) and 11th (bottom) groups at $t = 0.04$ s and 0.06 s.

with 274,625 vertices and 262,144 elements is used, and the corresponding nonlinear system has 3,020,875 unknowns (11 degrees of freedom per vertex).

In Table 1, we observe that the number of Newton and GMRES iterations remain approximately constant for both methods as the number of processors increases. The total compute time for the Gauss-Seidel sweep is comparable to that of the fully coupled RAS when the number of processors is small (48 and 96), but the Gauss-Seidel algorithm is slower (due to the improved concurrency of the RAS method) when more processors are used: 16% slower on 192 processors and 42% slower on 384. The average number of GMRES iterations required by the Gauss-Seidel sweep is a little smaller than that of the fully coupled RAS, but it does not compensate for the extra cost incurred by the sequential application of the preconditioner.

Table 1: A comparison between the fully coupled RAS and the Gauss-Seidel sweep. A nonlinear system with 3,020,875 unknowns is solved by the inexact Jacobian-free Newton-Krylov method. The “fully coupled RAS" here is a one-level fully coupled RAS.

	Fully coupled RAS				Gauss-Seidel sweep			
n_p	NI	LI	Time [s]	EFF [%]	NI	LI	Time [s]	EFF [%]
48	2.27	33.60	2258.70	100	2.36	27.50	2258.50	100
96	2.27	33.76	1229.30	92	2.36	28.64	1278.80	88
192	2.27	34.00	645.25	88	2.36	28.12	749.02	75
384	2.27	34.56	372.14	76	2.36	28.08	528.29	53

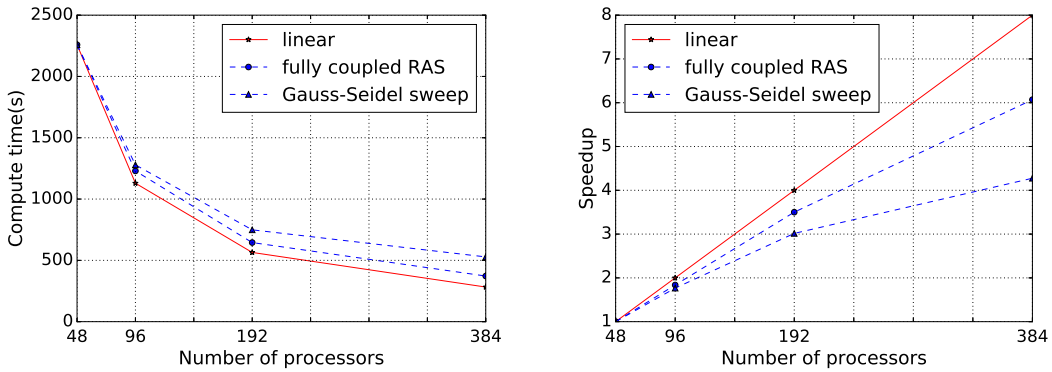

Figure 5: Total compute time (left) and speedup (right) using fully coupled RAS and Gauss-Seidel sweep. The “fully coupled RAS" here is a one-level fully coupled RAS

The parallel efficiency of the fully coupled RAS is 76%, while that of the Gauss-Seidel sweep drops to 53% when we use 384 processors. The total compute time and the corresponding speedup are also shown in Fig. 5. The compute time and speedup of the fully coupled RAS is always better than that of the Gauss-Seidel sweep in Fig. 5. The preconditioner consists of two stages; the setup and the application. Both stages are required to be scalable to make the overall algorithm scalable.

We show the time spent on the preconditioner setup and application in Figs. 6 and 7 for both the fully coupled RAS and the Gauss-Seidel sweep. In Fig. 6, we see that the setup of the fully coupled RAS is much cheaper than that of the Gauss-Seidel sweep for all processor counts, and the corresponding speedup is also much better. The fully coupled RAS is 5 times faster than that of the Gauss-Seidel sweep in terms of the time spent on the preconditioner setup. In Fig. 7, the application of the fully coupled RAS is 10 times faster than that of the Gauss-Seidel sweep, and it scales well as the number of processors increases.

4.2 Comparison with HYPRE BoomerAMG

HYPRE[9] is a library of linear solvers and preconditioners, and its “BoomerAMG” preconditioner has been used successfully in a wide range of application areas. In this section, we compare our proposed RAS approach to BoomerAMG

Figure 6: Compute time (left) and speedup (right) for preconditioner setup. The “fully coupled RAS" here is a one-level fully coupled RAS

Figure 7: Compute time (left) and speedup (right) for preconditioner application. The “fully coupled RAS" here is a one-level fully coupled RAS

for the same problem configuration discussed in the previous section. HYPRE's default BoomerAMG parameters are used, and the RAS parameters remain unchanged from the previous section. Both RAS and BoomerAMG are applied directly to the full preconditioning matrix. The numerical results are summarized in Table 2 and Figs. 8, 9, and 10.

As shown in Table 2, the number of Newton iterations is nearly identical for both cases as the number of processor cores is increased, but the RAS algorithm requires more linear iterations per Newton iteration. Despite this, the total computation time for RAS is less than BoomerAMG, especially for higher processor counts. The parallel efficiency of BoomerAMG drops to 52% at 384 cores, while RAS achieves a parallel efficiency of 76% at this scale. The superior nature of RAS can also be observed in Fig. 8, which compares the speedup of the two methods. On the left hand side of Fig. 8, the total compute time of RAS is always less than that of BoomerAMG as the number of processor cores is increased. Similarly, in Figs. 9 and 10, RAS is more efficient than BoomerAMG in terms of the preconditioner setup and application times. The setup and application phases of the RAS method both scale well with the number of processor cores, which is necessary to ensure that the overall algorithm remains efficient for large processor counts.

Table 2: A comparison between RAS and HYPRE's BoomerAMG preconditioner. A nonlinear system with 3,020,875 unknowns is solved by the inexact Jacobian-free Newton-Krylov method. The "fully coupled RAS" here is a one-level fully coupled RAS

	RAS				HYPRE BoomerAMG			
n_p	NI	LI	Time [s]	EFF [%]	NI	LI	Time [s]	EFF [%]
48	2.27	33.60	2258.70	100	2.36	18.73	2298	100
96	2.27	33.76	1229.30	92	2.36	18.73	1321.1	85
192	2.27	34.00	645.25	88	2.36	18.69	776.66	72
384	2.27	34.56	372.14	76	2.36	18.53	536.67	52

Figure 8: Total compute time (left) and speedup (right) using RAS and HYPRE. The "fully coupled RAS" here is a one-level fully coupled RAS

Figure 9: Compute time (left) and speedup (right) for preconditioner setup using RAS and HYPRE. The "fully coupled RAS" here is a one-level fully coupled RAS

4.3 Two-level method

The one-level method works well for relatively small-scale problems on moderate numbers of processors, but it deteriorates as the number of unknowns increases. To address this issue, we introduce one coarse space to construct the two-level method. We compare the performance of the two-level and one-level methods in this section. Both a "small" (3,020,875 unknowns) and "large" (23,613,579 unknowns) version of the problem are tested. For the small

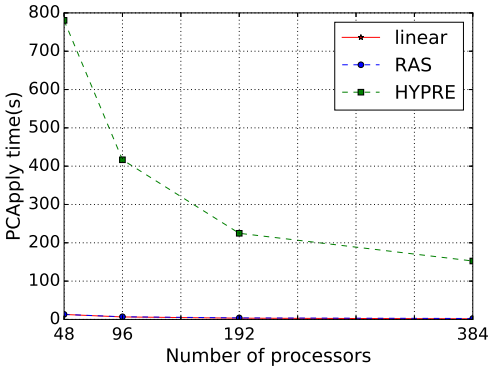

Figure 10: Compute time (left) and speedup (right) for preconditioner application using RAS and HYPRE. The “fully coupled RAS" here is a one-level fully coupled RAS

Table 3: Performance of the two-level method.

	One-level				Two-level			
n_p	NI	LI	Time [s]	EFF [%]	NI	LI	Time [s]	EFF [%]
				3,020,875 unknowns				
384	2.27	34.68	334.91	–	2.27	19.36	332.81	–
576	2.27	35.28	254.63	–	2.27	19.36	284.82	–
768	2.27	35.40	245.24	–	2.27	18.92	251.31	–
1152	2.27	35.96	320.00	–	2.27	18.72	270.40	–
				23,613,579 unknowns				
384	2.27	58.84	3916.6	100	2.27	19.24	2113.90	100
576	2.27	59.28	2679.3	97	2.27	19.24	1507.00	94
768	2.27	59.76	2077.7	94	2.27	19.28	1157.80	91
1152	2.27	60.16	1566.7	83	2.27	19.28	877.45	80

problem, the same mesh as in the previous test is used. The large problem employs a mesh with 2,146,689 vertices and 2,097,152 elements. The test is carried out using 384, 576, 768, and 1152 processors.

The results are summarized in Table 3, where we observe, in particular, that the required number of GMRES iterations is much higher for the one-level method than the two-level method. Slow growth in the number of linear iterations is required in order for a method to be considered scalable. The one- and two-level methods have similar performance for the small-scale problem in terms of the total compute time.

For the large-scale problem, the number of GMRES iterations for the two-level method is one-third that of the one-level method, and therefore the two-level method is twice as fast as the one-level method in terms of the total compute time. When we increase the number of processors, the number of GMRES iterations required for both algorithms remains approximately constant. Both algorithms are scalable with respect to the number of processors in terms of the total compute time and the number of GMRES iterations. We observed a parallel efficiency of around

Table 4: Impact of overlap size δ for the two-level method.

n_p	δ	NI	LI	Time [s]	EFF [%]
384	0	2.27	20.48	2174	100
384	1	2.27	19.24	2120	100
384	2	2.27	19.00	2131	100
576	0	2.27	20.92	1578	92
576	1	2.27	19.24	1507	94
576	2	2.27	19.20	1505	94
768	0	2.27	20.80	1193	91
768	1	2.27	19.28	1156	92
768	2	2.27	19.24	1153	92
1152	0	2.27	21.52	963	75
1152	1	2.27	19.28	922	77
1152	2	2.27	19.24	872	81

80% for both algorithms even when running on over a thousand processors. Finally, note that in Table 3, we do not report the parallel efficiency of the algorithms for the small-scale problem because in that case the problem is too small to measure efficiency in a meaningful way.

4.4 Impact of Schwarz parameters

The Schwarz algorithm has several important parameters, such as the overlap size δ, the ILU fill level, and the submatrix reordering, which affect the parallel performance of the method. We test these parameters in this subsection with the same configuration as above, and the results are summarized in Tables 4–6. In Table 4, for example, we see that the number of GMRES iterations slightly decreases and the number of Newton iterations stays constant as we increase the overlap size. The total compute time when using $\delta = 2$ is smaller than that obtained using $\delta = 0$ or $\delta = 1$ for all processor counts except 384, where the algorithm with $\delta = 1$ is slightly better. When we increase the number of processors, the average number of GMRES iterations for the $\delta = 0$ case increases slightly while the number of iterations for the $\delta = 1$ and 2 cases remain approximately constant. All cases have good scalability in terms of the compute time and the number of GMRES iterations.

Increasing the ILU fill level usually reduces the average number of GMRES iterations while increasing the cost of the subdomain solver. In Table 5, when we increase the fill level, we see that the number of GMRES iterations is indeed reduced, but the overall computation time is actually increased due to the extra complexity of the subdomain solver. ILU(0) has the best performance among all these cases except when $n_p = 1152$, where the compute time of ILU(1) is less than ILU(0) by about 36 seconds. Again, we observe that all cases have good scalability.

Degree of freedom reordering is used to reduce the bandwidth and improve the matrix nonzero structure. In this test, ILU(2) is used as the subdomain solver. In Table 6, we see that different reordering schemes result in different

Table 5: Impact of ILU fill level on the two-level method.

n_p	ILU	NI	LI	Time [s]	EFF [%]
384	ILU(0)	2.27	19.24	2120	100
384	ILU(1)	2.27	17.72	2166	100
384	ILU(2)	2.27	16.56	2594	100
576	ILU(0)	2.27	19.24	1507	94
576	ILU(1)	2.27	17.72	1528	95
576	ILU(2)	2.27	16.84	1839	94
768	ILU(0)	2.27	19.28	1156	92
768	ILU(1)	2.27	17.68	1166	94
768	ILU(2)	2.27	16.92	1437	90
1152	ILU(0)	2.27	19.28	922	77
1152	ILU(1)	2.27	17.84	886	81
1152	ILU(2)	2.27	17.28	1116	77

Table 6: Impact of submatrix reordering. ILU(2) is used as the subdomain solver, and the system of equations is solved with the two-level method while varying the reordering scheme.

n_p	Reordering	NI	LI	Time [s]	EFF [%]
384	RCM	2.27	16.08	2423.8	100
384	ND	2.27	16.52	2779.4	100
384	1WD	2.27	16.12	2411.8	100
384	QMD	2.27	16.60	3642.9	100
576	RCM	2.27	16.40	1757.7	92
576	ND	2.27	16.92	1932.8	96
576	1WD	2.27	16.32	1711.2	94
576	QMD	2.27	16.96	2395.5	> 100
768	RCM	2.27	16.36	1422.3	85
768	ND	2.27	16.88	1552.6	90
768	1WD	2.27	16.40	1327.6	90
768	QMD	2.27	16.92	1770.7	> 100
1152	RCM	2.27	16.84	1016.1	79
1152	ND	2.27	17.24	1124.8	82
1152	1WD	2.27	16.84	1020.3	79
1152	QMD	2.27	17.24	1268.7	96

performance characteristics. QMD is the worst for this particular problem because QMD was originally designed for symmetric matrices, and the matrices used in these tests are non-symmetric. RCM and 1WD have similar performance in terms of the total compute time and the number of GMRES iterations. The ND reordering requires almost the same number of iterations as QMD, but ND performs better than QMD in terms of the total compute time. For all cases we observe linear scalability, that is, the compute time is halved when we double the number of processors.

4.5 Temperature feedback

In this section, we solve the multigroup neutron diffusion equations in a transient scenario coupled with adiabatic heating (heat equation without conduction term) and temperature feedback to the nuclear cross sections. This example is also based on the TREAT, which is designed to expose nuclear fuel experiments to high-power transients: up to 18 GW, or about 6 times the power of a commercial light water reactor.

In TREAT, temperature feedback to the cross sections occurs through spectral shift, which decreases fission rates and increases leakage and absorption with increasing temperature. In this scenario, the power increases almost exponentially until the core reactivity is offset by spectral shift caused by the increase in temperature. After the reactor runs through its peak power, it settles on a new equilibrium core temperature and power level at which it will remain until further perturbation of its reactivity. We end the simulation sufficiently early that the effects of heat conduction can be ignored.

The simulation is run for $t \in (0, 2]$ with a mesh of 274,625 nodes and 262,144 elements. The resulting system of nonlinear equations has 3,020,875 unknowns in total. The entire simulation takes 4668.2 s on 768 processor cores. The variation of the temperature and the power with time is shown in Fig. 11, where "Averaged" represents the average temperature over the whole computational domain, and "Maximum" represents the maximum temperature point in the computational domain. In the graph on the right hand side of Fig. 11, "power" is defined as $\int_\Omega \rho_{\text{power}} \, \mathrm{d}x$.

We observe that the power grows rapidly and reaches a peak at about $t = 1.2$ s, and then rapidly decreases for $t \in [1.2, 2]$. The temperature increases slowly for $t \in [0., 0.5]$, rapidly for $t \in [0.5, 1.5]$, and slowly again for $t \in [1.5, 2]$. This complicated and realistic physical configuration challenges the nonlinear and linear solvers. The performance of the algorithm (in terms of the average number of Newton and GMRES iterations required at each time step) is shown in Fig. 12.

The number of required Newton iterations remains almost constant throughout the simulation. Two Newton iterations are required for $t \in [0, 0.7] \cup [1.6, 2]$ and three Newton iterations are required for $t \in [0.7, 1.6]$. The trend transition occurs for $t \in [0.7, 1.6]$, and therefore the resulting nonlinear system is expected to be more difficult to solve during that time period. The average number of linear iterations per Newton step also remains close to constant during the entire simulation.

The data of Fig. 12 demonstrate that the proposed algorithm is robust in the context of a realistic simulation with temperature feedback. The strong scalability of the algorithm is shown in Table 7. The mesh used for the strong scalability study consists of 2,146,689 nodes, 2,097,152 elements, and 23,613,579 unknowns. In Table 7, we observe that both the Newton and GMRES iteration counts remain approximately constant as we increase the number of processors, and the compute time is almost halved when we double the number of processor cores. The algorithm exhibits a parallel efficiency of around 70% on 1152 processors.

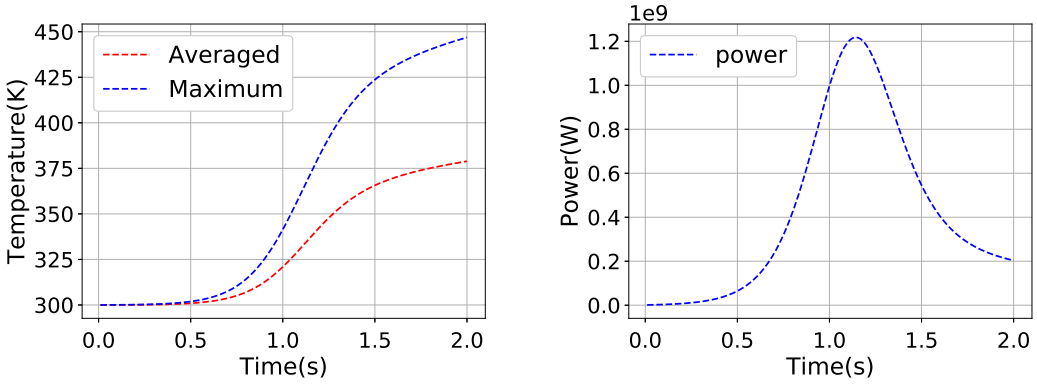

Figure 11: Temperature and power feedback for $t \in [0, 2]$.

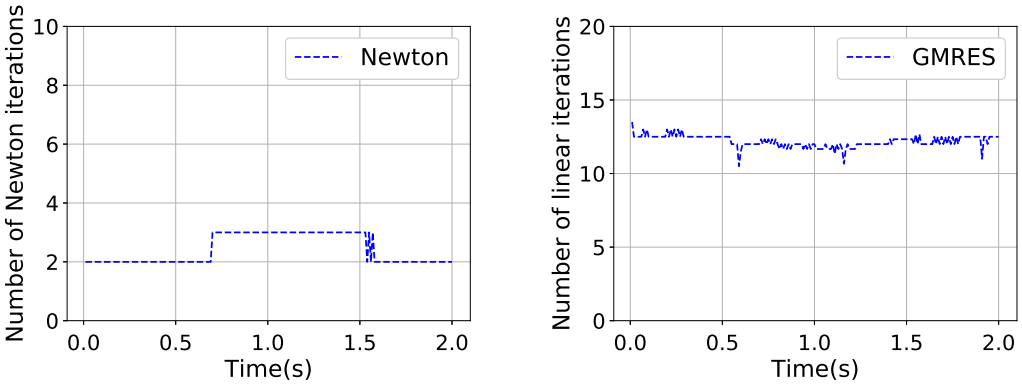

Figure 12: Number of the Newton and linear iterations for $t \in [0, 2]$ with the problem with temperature feedback.

In a multilevel method, the size of coarse space should remain small to ensure that its computation remains inexpensive, but if it is too small, the resulting algorithm will require too many linear iterations. Good scalability also requires that the coarse space dimension not change much as the number of processors increases. In Table 7, "rows$_c$" represents the number of coarse matrix rows, and "nonzeros$_c$" denotes the number of coarse matrix nonzeros. We observe that the ratio of rows$_c$ to fine matrix rows is around 0.6%, and the ratio of nonzeros$_c$ to fine matrix nonzeros is 0.8%. The size of the coarse space remains approximately constant as we increase the number of subdomains, which is an indication that the coarsening algorithm works effectively. Overall, the proposed algorithms are found to be robust and scalable for the temperature feedback application.

5 Conclusions

An implicit Newton-Krylov method based on a fully coupled and scalable two-level Schwarz preconditioner was studied for a challenging system of equations applicable to nuclear engineering analysis. This system was derived from the discretization of the transient multigroup neutron diffusion equations by a standard finite element method

Table 7: Strong scaling of the algorithm for the temperature feedback problem. The number of unknowns is 23,613,579 and the number of nonzeros in the matrix is 1,711,998,750 at the fine level.

n_p	NI	LI	Time [s]	rows_c	nonzeros_c	EFF [%]
192	2.27	17.16	4139.5	135,541	13,421,921	100
384	2.27	17.72	2283.9	135,965	13,512,525	91
576	2.27	17.72	1601.5	135,448	13,470,818	86
768	2.27	17.68	1246.3	135,156	13,420,703	83
1152	2.27	17.84	990.7	134,779	13,311,952	70

in space combined with an implicit time discretization. The coarse space (based on smoothed aggregation) was introduced to construct the two-level version of a Schwarz preconditioner. The preconditioner is applied in a fully coupled way so that all group variables are computed simultaneously. It was shown that the fully coupled method has better performance than the block Gauss-Seidel sweep algorithm, and that the two-level method further improves the parallel performance as the number of processors increases. The scalability of the proposed algorithm, in terms of the total compute time and the number of nonlinear and linear iterations, was also demonstrated.

Acknowledgments

This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

Appendix

This appendix contains tabulated data associated with the material coefficients in the multigroup neutron diffusion equations.

Table 8: Material properties of the graphite domain at $T = 293.6$ K. Since there are no nuclear reactions occurring on the graphite, material parameters for the fission term and the DNP term are zero. There are, however, still scattering events in the graphite, and we show those parameters in a different table. The density of the nuclear fuel is $\rho = 1.52639336$ gcm^{-3}, and the density of the graphite is $\rho = 0$.

g	v_g [cm s^{-1}]	D_g [cm]	$\Sigma_{r,g}$ [cm^{-1}]
1	2.959280e+9	4.371631e0	3.390080e−2
2	9.412739e+8	2.184245e0	1.161440e−2
3	1.731065e+8	1.321336e0	1.211170e−2
4	3.238520e+7	1.304055e0	1.300950e−2
5	7.313684e+6	1.300991e0	1.516310e−2
6	1.920953e+6	1.300082e0	1.622090e−2
7	8.049456e+5	1.293077e0	3.782370e−2
8	4.583182e+5	1.266950e0	3.890520e−2
9	3.380228e+5	1.216505e0	6.807460e−2
10	2.471632e+5	1.210840e0	5.145000e−2
11	1.396748e+5	1.196609e0	4.873540e−2

Table 9: Scattering parameters, $\Sigma_{s,g'\to g}$ [cm^{-1}], on the graphite domain at $T = 293.6$ K.

$g' \to g$	1	2	3	4	5	6	7	8	9	10	11
1	6.690920e−2										
2	3.351650e−2	1.746300e−1									
3	6.111820e−5	1.161090e−2	2.560990e−1								
4	1.500860e−7	1.340240e−8	1.210080e−2	2.576820e−1							
5				1.300180e−2	2.561190e−1	3.507450e−6					
6					1.513240e−2	2.552350e−1	1.022770e−3				
7						1.606780e−2	2.346800e−1	2.613600e−3	3.353300e−5	2.350210e−6	
8						2.869120e−5	3.539410e−2	2.363250e−1	2.694170e−2	4.042030e−3	8.219040e−4
9							7.808310e−4	2.864410e−2	2.129410e−1	2.565280e−2	4.275910e−3
10							3.053970e−4	6.505420e−3	3.725210e−2	2.325990e−1	4.200760e−2
11							3.713940e−5	6.413770e−4	3.169930e−3	2.082870e−2	2.273720e−1

Table 10: Material properties for nuclear fuel at $T = 293.6$ K. The scattering parameters of the nuclear fuel are shown in a different table. Note that in this computation, for a given group, $\chi_{d,i,g} = \chi_{d,j,g}$, but in general, $\chi_{d,i,g} \neq \chi_{d,j,g}$.

g	v_g [$\mathrm{cm\,s^{-1}}$]	D_g [cm]	$\Sigma_{r,g}$ [$\mathrm{cm^{-1}}$]	$\nu\Sigma_{f,g}$ [$\mathrm{cm^{-1}}$]	$\kappa\Sigma_{f,g}$ [$\mathrm{J\,cm^{-1}}$]	$\chi_{d,i,g}$	$\chi_{p,g}$
1	2.949626e+9	3.171522e0	4.799060e−2	2.845520e−5	3.025787e−16	0.000000e0	1.731860e−1
2	9.821543e+8	1.700550e0	1.338930e−2	2.416280e−5	3.088935e−16	8.755640e−1	8.117770e−1
3	1.768128e+8	9.838443e−1	1.534450e−2	4.696380e−5	6.250405e−16	1.224150e−1	1.495740e−2
4	3.269320e+7	9.610330e−1	1.716630e−2	2.185340e−4	2.905171e−15	1.941470e−3	7.913460e−5
5	7.359868e+6	9.608252e−1	2.039280e−2	8.303450e−4	1.102883e−14	7.406730e−5	5.335540e−7
6	1.929057e+6	9.607034e−1	2.207510e−2	5.807100e−4	7.710701e−15	4.342090e−6	9.490130e−9
7	8.085774e+5	9.547515e−1	5.391220e−2	2.610080e−3	3.465703e−14	5.828650e−7	0.000000e0
8	4.619450e+5	9.384675e−1	6.345620e−2	4.400400e−3	5.842866e−14	0.000000e0	0.000000e0
9	3.382927e+5	9.075703e−1	1.192890e−1	6.558340e−3	8.708196e−14	0.000000e0	0.000000e0
10	2.475413e+5	8.973237e−1	9.463540e−2	9.526350e−3	1.264916e−13	3.104860e−7	0.000000e0
11	1.403660e+5	8.604660e−1	9.477190e−2	1.757570e−2	2.333717e−13	0.000000e0	0.000000e0

Table 11: Material properties for delayed neutron precursors for the nuclear fuel at $T = 293.6$ K.

	λ_i [$\mathrm{s^{-1}}$]	β_i
1	1.333600e−2	2.403820e−4
2	3.273900e−2	1.247060e−3
3	1.207800e−1	1.193690e−3
4	3.027810e−1	2.654130e−3
5	8.494920e−1	1.092640e−3
6	2.853010e0	4.587350e−4

Table 12: Scattering parameters, $\Sigma_{s,g'\to g}$ [cm^{-1}], for the nuclear fuel at $T = 293.6$ K.

$g' \to g$	1	2	3	4	5	6	7	8	9	10	11
1	8.887180e−2										
2	4.750900e−2	2.254670e−1									
3	9.214310e−5	1.336460e−2	3.450800e−1								
4	2.668750e−7	2.421400e−8	1.526740e−2	3.499310e−1							
5				1.687780e−2	3.467390e−1	5.288770e−6					
6					1.978050e−2	3.450260e−1	1.498180e−3				
7						2.151340e−2	3.150300e−1	4.537920e−3	3.259550e−5	2.305710e−6	
8						2.623450e−5	4.980090e−2	3.091960e−1	4.546040e−2	6.402610e−3	9.460720e−4
9							7.374000e−4	4.594210e−2	2.605940e−1	4.516300e−2	6.960650e−3
10							2.738180e−4	9.647840e−3	6.490140e−2	2.909420e−1	7.671730e−2
11							3.346000e−5	7.174800e−4	5.096190e−3	3.758930e−2	2.957840e−1

Table 13: Some of the temperature-dependent properties of the nuclear fuel. Other fuel and graphite material properties, which have a similar temperature dependence, are not shown here.

T [K]	D_{11} [cm]	$\nu\Sigma_{f,11}$ [cm^{-1}]	$\Sigma_{s,1\to2}$ [cm^{-1}]
2.936000e+2	8.604660e−1	1.757570e−2	4.750900e−2
4.000000e+2	8.388551e−1	1.735170e−2	4.750600e−2
6.000000e+2	8.004085e−1	1.709310e−2	4.748140e−2
8.000000e+2	7.660000e−1	1.691230e−2	4.748830e−2

References

1. Adams MF. Algebraic multigrid methods for direct frequency response analyses in solid mechanics. Comput Mech. 2007;**39**(4): 497–507.

2. Aragonés JM, Ahnert C, and Garcia-Herranz N. The analytic coarse-mesh finite difference method for multigroup and multidimensional diffusion calculations. Nucl Sci Eng. 2007;**157**(1): 1–15.

3. Baker BA, Ortensi J, and DeHart MD. 2016. *FY 2016 Status Report on the Modeling of the M8 Calibration Series using MAMMOTH*, Technical Report INL/EXT–16-40023, Idaho National Laboratory.

4. Balay S, Abhyankar S, Adams MF., Brown J, Brune P, Buschelman K, Dalcin L, Eijkhout V, Gropp WD., Kaushik D, Knepley MG., McInnes LC, Rupp K, Smith BF, Zampini S, Zhang H, and Zhang H. 2016. *PETSc Users Manual*, Technical Report ANL-95/11 - Revision 3.7, Argonne National Laboratory.

5. Barrault M, Lathuiliere B, Ramet P, and Roman J. Efficient parallel resolution of the simplified transport equations in mixed-dual formulation. J Comput Phys. 2011;**230**(5): 2004–2020.

6. Cai XC and Sarkis M. A restricted additive Schwarz preconditioner for general sparse linear systems. SIAM J Sci Comput. 1999;**21**(2): 792–797.

7. Dennis Jr JE and Schnabel RB. Numerical Methods for Unconstrained Optimization and Nonlinear Equations: SIAM; 1996.

8. Duderstadt JJ and Hamilton LJ. Nuclear Reactor Analysis: Wiley; 1976.

9. Falgout RD and Yang UM. Hypre: A library of high performance preconditioners, International conference on computational science; 2002. p. 632–641.

10. Gaston DR, Permann CJ, Peterson JW, Slaughter AE, Andrš D, Wang Y, Short MP, Perez DM, Tonks MR, Ortensi J, and Martineau RC. Physics-based multiscale coupling for full core nuclear reactor simulation. Ann Nucl Eng. October 2015;**84**: 45–54.

11. Jamelot E and Ciarlet P. Fast non-overlapping Schwarz domain decomposition methods for solving the neutron diffusion equation. J Comput Phys. 2013;**241**: 445–463.

12. Kaper HG, Leaf GK, and Lindeman AJ. A timing comparison study for some high order finite element approximation procedures and a low order finite difference approximation procedure for the numerical solution of the multigroup neutron diffusion equation. Nucl Sci Eng. 1972;**49**(1): 27–48.

13. Kirk BS, Peterson JW, Stogner RH, and Carey GF. libMesh: A C++ library for parallel adaptive mesh refinement/coarsening simulations. Eng Comput. 2006;**22**(3–4): 237–254.

14. Knoll DA and Keyes DE. Jacobian-free Newton-Krylov methods: a survey of approaches and applications. J Comput Phys. 2004;**193**(2): 357–397.

15. Knoll DA, Park H, and Newman C. Acceleration of k-eigenvalue/criticality calculations using the Jacobian-free Newton-Krylov method. Nucl Sci Eng. 2011;**167**(2): 133–140.

16. Kong F and Cai XC. A highly scalable multilevel Schwarz method with boundary geometry preserving coarse spaces for 3D elasticity problems on domains with complex geometry. SIAM J Sci Comput. 2016;**38**(2): C73–C95.

17. Kong F and Cai XC. Scalability study of an implicit solver for coupled fluid-structure interaction problems on unstructured meshes in 3D. Int J High Perform Comput Appl. 2016.

18. Kong F and Cai XC. A scalable nonlinear fluid–structure interaction solver based on a Schwarz preconditioner with isogeometric unstructured coarse spaces in 3D. J Comput Phys. 2017;**340**: 498–518.

19. Lewis EE and Miller WF. Computational Methods of Neutron Transport: John Wiley and Sons; 1984.

20. Ortensi J, DeHart MD, G Frederick N, Wang Y, Alberti AL, and Palmer TS. 2015. *Full Core TREAT Kinetics Demonstration Using Rattlesnake/BISON Coupling Within MAMMOTH*, Technical Report INL/EXT–15-36268, Idaho National Laboratory.

21. Purwadi MD, Tsuji M, Narita M, and Itagaki M. A hierarchical domain decomposition boundary element method applied to the multiregion problems of neutron diffusion equations. Nucl Sci Eng. 1998;**129**(1): 88–96.

22. Quarteroni A and Valli A. Domain Decomposition Methods for Partial Differential Equations: Oxford University Press; 1999.

23. Saad Y. Iterative Methods for Sparse Linear Systems: SIAM; 2003.

24. Saad Y. Numerical Methods for Large Eigenvalue Problems: Revised Edition: SIAM; 2011.

25. Saad Y and Schultz MH. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J Sci Stat Comput. 1986;**7**(3): 856–869.

26. Scheichl R. Parallel solvers for the transient multigroup neutron diffusion equations. Int J Numer Methods Eng. 2000;**47**(10): 1751–1771.

27. Semenza LA, Lewis EE, and Rossow EC. The application of the finite element method to the multigroup neutron diffusion equation. Nucl Sci Eng. 1972;**47**(3): 302–310.

28. Smith B, Bjorstad P, and Gropp WD. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations: Cambridge University Press; 2004.

29. Toselli A and Widlund OB. Domain Decomposition Methods: Algorithms and Theory, Vol. 34: Springer; 2005.

30. Vaněk P, Mandel J, and Brezina M. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. Comput. 1996;**56**(3): 179–196.

31. Vidal-Ferràndiz A, González-Pintor S, Ginestar D, Verdú G, and Demazière C. Schwarz type preconditioners for the neutron diffusion equation. J Comput Appl Math. 2017;**309**: 563–574.

32. Wang Y, Bangerth W, and Ragusa J. Three-dimensional *h*-adaptivity for the multigroup neutron diffusion equations. Prog Nucl Eng. 2009;**51**(3): 543–555.

33. Wang Y, Schunert S, DeHart M, Martineau R, and Zheng W. Hybrid PN-SN with Lagrange multiplier and upwinding for the multiscale transport capability in Rattlesnake. Prog Nucl Eng. 2017.

□