INL/RPT-24-80749

NEAMS Technical Area Support in MOOSE

September 2024

Nuclear Energy Advanced Modeling and Simulation M3 Milestone September 2024

Guillaume Giudicelli[1], Mengnan Li[1], Logan Harbour[1], Patrick Behne[1], Oana Marin[1], Roy Stogner[1], Cody Permann[1], and Alexander Lindsay[1]

[1]*COMPUTATIONAL FRAMEWORKS*

Idaho National Laboratory

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

INL/RPT-24-80749

NEAMS Technical Area Support in MOOSE

Nuclear Energy Advanced Modeling and Simulation M3 Milestone
September 2024

Guillaume Giudicelli[1], Mengnan Li[1], Logan Harbour[1], Patrick Behne[1], Oana Marin[1], Roy Stogner[1], Cody Permann[1], and Alexander Lindsay[1]

[1]COMPUTATIONAL FRAMEWORKS

September 2024

Idaho National Laboratory
Computational Frameworks
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Page intentionally left blank

Page intentionally left blank

CONTENTS

FIGURES

ACRONYMS

AD	Automatic Differentiation
API	Application Programming Interface
HTGR	High Temperature Gas Reactor
LSC	Least Squares Commutator
MOOSE	Multiphysics Object-Oriented Simulation Environment
MR	Merge Request
NEAMS	Nuclear Energy Advanced Modeling and Simulation
PETSc	Portable Extensible Toolkit for Scientific Computation
PR	Pull Request
SLEPc	Scalable Library for Eigenvalue Problem Computations
TA	Technical Area

Page intentionally left blank

1. EXECUTIVE SUMMARY

The Multiphysics Object-Oriented Simulation Environment (MOOSE) framework is a foundational capability used by the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program to create over 15 different simulation tools for advanced nuclear reactors. Due to this ubiquity, improvements to the framework in support of modeling and simulation goals are critical to the program. These improvements can take many forms, including optimization, improved user experience, streamlined application programming interfaces (APIs), parallelism, and other new capabilities. The work described in this report was conducted in direct support of the simulation tools and has already been deployed. The capabilities outlined in this report include enabling selective polynomial basis refinement, implementing a custom convergence system, building a scalable preconditioner for saddle-point problems, and much more.

2. INTRODUCTION

The NEAMS program aims to develop simulation tools in support of the nuclear industry. These tools are meant to accelerate reactor design, licensing, demonstration, and deployment. The program is split into five Technical Areas (TAs): Fuel Performance, Thermal Fluids, Structural Materials and Chemistry, Reactor Physics, and Multiphysics Applications. The "physics" TAs are responsible for delivering simulation tools to vendors, laboratories, and licensing authorities, while the Multiphysics Applications TA creates foundational capabilities for the program and exercises the tools to test them and ensure their usability for the intended purpose.

Reactors are inherently multiphysical: heat conduction, neutronics, solid mechanics, fluid flow, chemistry, and material evolution all combine to create a complex system that needs to be simulated. To tackle that problem, the NEAMS program utilizes the MOOSE platform [1] to develop interoperable physics applications. Over 15 MOOSE-based physics applications are being developed within the program, with at least one in every TA. Each physics application focuses on a particular aspect of reactor simulation (e.g., BISON [2] for nuclear fuel performance and Griffin [3] for neutronics).

It is therefore critical to the program that MOOSE continues to be enhanced and supported. Capabilities and optimizations made to the framework are instantly available to all the NEAMS

applications, making for a large return on investment.

2.1 MOOSE

The MOOSE platform enables rapid production of massively parallel, multiscale, multiphysics simulation tools based on finite-element, finite-volume, and discrete ordinate discretizations. The platform was developed as open-source software on GitHub [4] and utilizes the lesser general public license (LGPL) 2.1, which allows for a large amount of flexibility. It is an active project, with dozens of code modifications merged weekly.

The core of the platform is a pluggable C++ framework that enables scientists and engineers to specify all the details of their simulations. Certain interfaces include: finite-element/finite-volume terms, boundary conditions, material properties, initial conditions, and point sources. By modularizing numerical simulation tools, MOOSE allows for an enormous amount of reuse and flexibility.

Beyond the core framework, the MOOSE platform also provides myriad supporting technologies for application development. This includes a build system, a testing system for both regression and unit testing, an automatic documentation system, visualization tools, and many physics modules. The physics modules are a set of common physics utilizable by application developers. Some of the more important modules are the solid mechanics, heat conduction, fluid flow, chemistry, and phase-field modules. All of this automation and reuse accelerates application development within the NEAMS program. In the following sections, we outline changes made to the MOOSE framework and its modules in order to support the various TAs.

3. GENERAL SUPPORT

3.1 Convergence System

Convergence of simulations in MOOSE is generally assessed by examining the norm of the residual of the nonlinear equations. This criterion is used for regular nonlinear solves and for examining the convergence of fixed point iterations of two simulations using `MultiApps`. Both a criteria on the absolute value of the norm and on the relative value with regards to the initial (before the solve) residual can be used.

With the variety of applications leveraging MOOSE came a wide number of needs, and the options for evaluating convergence quickly grew. Fixed point iterations and nonlinear solve convergence could be evaluated using `Postprocessors` instead. This was used, for example, to evaluate the mass imbalance in thermal hydraulics domain-overlapping coupling iterations [5], for the evolution of the k_{eff} as a convergence criterion for neutronics eigenvalue calculations or the control rod positions to decide the convergence of a neutronics criticality search. In solid mechanics calculations, it was observed that the residual for a reference solution, rather than the initial solution, was a better number to compare to when assessing the nonlinear residual convergence. The diversity of needs forces some of the major classes in MOOSE, such as the default `FEProblem` for nonlinear solves, to hold a lot of extra information and data to implement them. This code is partially duplicated in the `FixedPointSolve` class, which handles the iterations between `MultiApps`. Having to modify these central classes does not let users easily implement new convergence criteria, unless they can fit them inside a `Postprocessor`. The `Postprocessor` options did offer some generality in the user selection of a convergence criterion.

Considering there are many nested solves and iterative processes in MOOSE that require specific convergence criteria, as well as user requests to implement moving convergence criteria during simulations (gathered during the NEAMS-TH workshop in 2022), the `Convergence` system was created. `Convergence` objects are created by the user and can be retrieved by any solver class. The following objects were created:

- `ResidualConvergence` decides the convergence based on the norm of the residual of the nonlinear equations. This is the "classic" MOOSE convergence test
- `ReferenceResidualConvergence` decides the convergence based on the norm of the residual relative to the norm of a reference residual vector specified by the user
- `AugmentedLagrangianConvergence` decides the convergence of contact problems using the augmented Lagrangian method.

The following objects were conceptualized but not implemented yet:

- `SolutionDifferenceConvergence` decides convergence based on the relative difference in the solution vector, rather than the residual. This is notably useful to detect the occurrence

of a steady state when running relaxation transients.

- `PostprocessorConvergence` decides convergence based on the value of a postprocessor.
- `FixedIterationsConvergence` decides convergence based on the number of iterations.
- `ParsedComboConversion` decides convergence based on the evaluation of a parsed expression of the convergence decisions of other `Convergence` objects. This could notably be used to switch between different convergence objects at various times in the simulation.

Note that the `Convergence` system is not merged in MOOSE as of September 2024.

3.2 Miscellaneous Items

- Pull Request (PR) #26562: Allowed more complicated absolute paths in the `file_base` outputs parameter.
- PR #26936: Console objects have a parameter '`output_file`' that, when true, writes console output to a file. Parallel simulations would write information from all processors to the same file, which can lead to bad behavior. This PR introduced a patch that only writes console content from the head processor to the file. In the case that '`--keep-cout`' is passed to the executable, each processor writes its console content to a separate file.
- PR #27172: Prior to this PR, steady state detection with '`check_aux = true`' used the relative difference in the L2 norm of the *aggregated* auxvariables to determine whether the problem had reached steady state. This approach is problematic for simulations where one auxvariable changes much more quickly than others. In these cases, steady state will be falsely detected before the more quickly changing auxvariables stop changing. This PR introduced a solution to this issue. Instead of computing a relative difference norm aggregated over all auxvariables, this relative difference norm is computed separately for each auxvariable. Steady state is then only declared if the relative difference norm for *each* auxvariable is below the specified tolerance.
- PR #27239: This PR added an informative warning: when the '`disable_fixed_point_residual_norm_check`' is set to true and the user sets one of

three parameters, a warning is printed stating that these parameters will be ignored because the norm check is disabled. The three parameters are: '`fixed_point_rel_tol`', '`fixed_point_abs_tol`', and '`fixed_point_force_norms`'.

- PR #27240: This PR added the following features to the wall time interval-based checkpoint system: an easy way to turn off automatic checkpointing, outputting of checkpoint information to console, a new integration test to verify that subApps are not auto-checkpointed, a new regression test for minimum wall time (where checkpoint write time exceeds a wall time interval), an adjustment to the default wall time interval from 10 to 60 minutes, and changed checkpoint format from binary to compressed ASCII for compatibility with systems lacking XDR support.

- PR #27273: Fixed bugs in visualization output of Lagrange data on Tri7, in DirichletBoundary application on nodesets generated from sidesets, and in FEMContext quadrature selection for cases where the first variable of a system is higher-order than any of the variables being queried by the context. This last fix prevents excessive numbers of quadrature points from being used in some MOOSE GeneralField transfer operations.

- PR #27287: Users reported segmentation fault crashes from a subtly invalid input file. Though the input in this case was invalid, a segmentation fault or any crash was an unacceptable outcome. MOOSE now tests for the validity of the particular input file options used, and in invalid cases exits cleanly with a reasonable error message.

- PR #27330: This PR patched the bug in which the `MultiAppVectorPostprocessorTransfer` gave incorrect results when postprocessor values were transferred from subapps distributed over more than one processor to the parent app.

- PR #27422: Set PETSc's '`options_left`' option to false for `mesh-only` mode. Doing this keeps PETSc from emitting irrelevant warnings when running MOOSE in mesh-only mode.

- PR #27477: This MOOSE PR used functionality, newly added in libMesh PR #3736, to add valid MOOSE command line parameters to a "registered-as-used list", which is then used to notify PETSc so they are not interpreted by PETSc as unused parameters, which

could previously lead PETSc to raise confusing and invalid warnings at program exit. This implementation has the added benefit that any invalid command line parameters are still passed to PETSc, which then prints a valid warning for users.

- PR #27603: Users can choose to run a MOOSE input without solving by setting '`Problem/solve=false`'. When this is done, however, PETSc sees options that are used during the solve as 'unused', and emits a warning. Since the user intentionally instructs MOOSE not to solve, these warnings are irrelevant and should not be displayed to the user. PETSc's '`options_left`' parameter was set to false when '`Problem/solve=false`'. This suppresses the warning.
- PR #28392: Remove the execute_on parameter from Functions. As functions are evaluated on-the-fly, they are not intended to be executed and cache a value until the next execution.

4. PRONGHORN SUPPORT

Please refer to [6] and [7] for more information on the Pronghorn code.

4.1 Computing the Residual and Jacobian Together

In the last couple years, MOOSE added the capability to perform simultaneous calculation of the residual vector and Jacobian matrix. This capability is particularly advantageous for simulations that use Automatic Differentiation (AD), since the value and derivatives array present in the dual numbers used in AD represent the residual and Jacobian, respectively. Pronghorn heavily leverages AD for its finite volume simulations, so it was a strong candidate for leveraging the simultaneous calculation capability. PR #24865 made simultaneous residual and Jacobian calculation the default for Pronghorn simulations relying on the Navier-Stokes finite volume action for problem setup. This change brought an overall speedup of 15%, as shown when comparing the profiling graphs in Figure 1 and Figure 2.

4.2 Better Friction Formulations

PR #21230 made the definitions of Darcy and Forchheimer coefficients used in Pronghorn consistent with what is used most widely in literature. In particular, the friction force due to Darcy

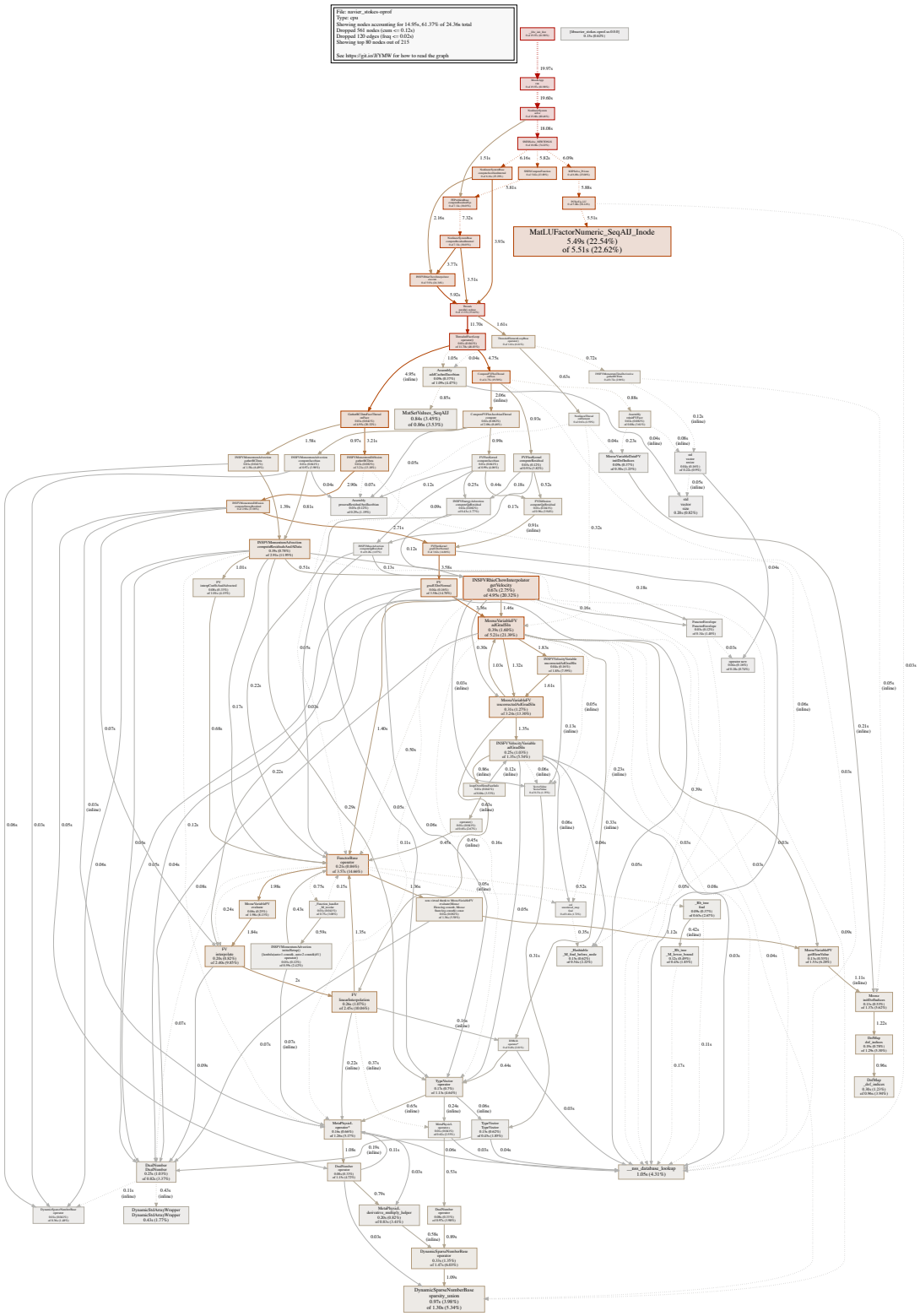


Figure 1: Profiling graph of a Pronghorn simulation with separate calculation of the residual and Jacobian.

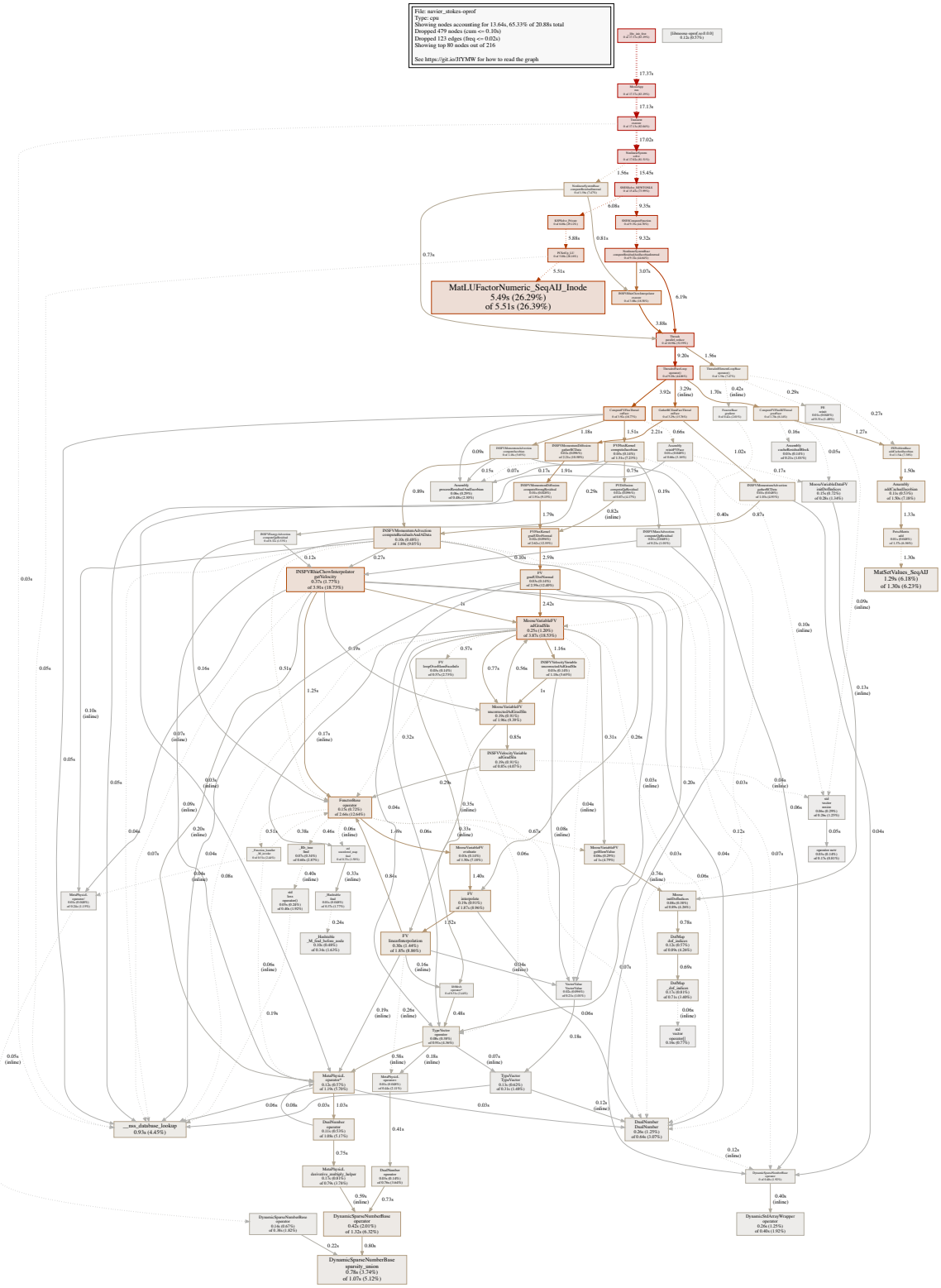


Figure 2: Profiling graph of a Pronghorn simulation with simultaneous calculation of the residual and Jacobian.

friction is now described by $f_D \epsilon \mu v$, where f_D is the Darcy coefficient, μ is the dynamic viscosity, ϵ is the porosity, and v is the interstitial velocity; the Forchheimer friction is described by $f_F \epsilon \frac{\rho v^2}{2}$, where f_F is the Forchheimer friction coefficient and ρ is the density.

4.3 Least Squares Commutator Preconditioning

Exploration of field split preconditioning using the Least Squares Commutator (LSC) was begun as a thermal hydraulics area task in fiscal year '23 and was finished using the technical area support task this year. LSC is outlined in [8] and [9]. It was implemented in MOOSE through the Portable Extensible Toolkit for Scientific Computation (PETSc) Merge Request (MR) #6642 and PR #24883. Using LSC, we were able to solve a 70 million degree of freedom problem on Sawtooth using 3,504 processes in 279 seconds for a Reynolds number of 1.

4.4 Support for Spline-Based Table Lookup Fluid Properties

Spline-based table lookup [10] fluid properties were implemented for several fluids, including air, nitrogen, carbon dioxide, and helium, originally to support RELAP-7 work [11]. These fluid properties are implemented quite extensively to support look-ups with both the conservative (specific volume, specific internal energy) and primitive (pressure, temperature) variables. However, a specific routine was not implemented in all these properties, and it is needed to compute the Jacobian contribution of the time derivative term in a weakly-compressible formulation. This routine is shown below.

```
void
rho_from_p_T(const ADReal & p,
             const ADReal & T,
             ADReal & rho,
             ADReal & drho_dp,
             ADReal & drho_dT);
```

We modified the Navier Stokes module to offer an option to avoid using this specific fluid property routine, and instead neglect the partial derivatives with respect to individual system degrees of freedom of the partial derivatives of density with respect to pressure and temperature.

4.5 Restore Transient MultiApps by Default in All Cases

In a transient simulation, both the parent and child applications take time steps. Depending on the user-selected order of execution, the parent app may go from the beginning to the end of its time step before or after the child applications. If sub-cycling is activated, the child application may take more time steps than the parent to reach the end of the parent time step.

Because fixed point iterations may be performed between the parent and child applications, the child applications save a checkpoint (a backup) at the beginning of the time step. This checkpoint is used to restart the time step on the next fixed point iteration.

Prior to this work, transient `MultiApps` were set to not restore to their initial state unless either sub-cycling, catch-up, or any other advanced modes were active. In their default mode, they would start their time step from the previous solution. This is not usually a problem, rather it is an optimization, as we can skip the solves or start from a very good initial guess, unless the simulation has states. The state at the end of the time step is not the same as the state at the beginning, so the simulation would essentially take another time step on top of the one it took on the previous fixed point iteration. Unfortunately, the domain-overlapping coupling [5] work did not use any advanced modes, and had states due to computation of coupling quantities at the beginning of the time step. Hence, it was affected by this issue. The problem was corrected by making the option not to restore a user parameter, which only expert users attempting to optimize a fixed point iteration scheme should use.

4.6 Boundary Ghosting

MOOSE natively supports distributed memory parallelism using MPI. This support is possible thanks to both libMesh and PETSc supporting distributed memory parallelism. There are numerous utilities in libMesh to facilitate parallel communications, notably the Templated Interface to MPI (TIMPI). The solution vectors and numerous data structures, such as the mesh, can be distributed across each process. Distributing the data keeps the memory cost reasonable for each process.

Processes must communicate with other processes for a number of reasons. For example, the additive Schwartz method is commonly used for preconditioning, and the blocks created by the

method overlap by one row. This overlap must be communicated between domains. Another example is the computation of gradients in finite volume discretizations, which involve stencils spanning several elements. At the boundary of the domain of a process, some of the elements are on other processes' domains. The values must be communicated between domains to compute the gradient. In MOOSE, we refer to the process of sharing of information as 'ghosting.' There are three levels of ghosting: coupling, algebraic, and geometric.

Geometric ghosting means that a process needs information about the element geometry from elements belonging to other processes. Algebraic ghosting means that a process needs information about the variable values from elements belonging to other processes. Coupling ghosting means that a process needs information about the sparsity pattern (e.g., the coupling between DoFs) from elements belonging to other processes. Coupling ghosting implies algebraic, which implies geometric ghosting. For the finite volume stencil problem, the ghosting required is algebraic, and extends a few layers (two without skewness correction, three with) around the domain of each process.

Ghosting needs are quite varied, and the `RelationshipManager` system was added to handle them. In fluid dynamics simulations, it is often necessary to know the distance from a cell to the wall for the purpose of modeling turbulence. The mixing length model, for example, determines the turbulent viscosity from the distance to the wall. In this case, each process needs to know the shortest distance from each of their cells to a given set of boundaries. In a distributed context, the process does not own the entire set of boundaries; it may not even own a single cell near this boundary. With geometric ghosting of the boundary, achieved using the `GhostBoundary` `RelationshipManager`, each process can compute wall distance from each of its owned elements to the boundary.

5. GRIFFIN SUPPORT

Please refer to [12] for more information on Griffin.

5.1 Custom Eigenvalue Normalization for SLEPc

PR #25204 introduced the ability to perform custom normalizations for eigenvalue calculations in MOOSE. This work was performed because Griffin had observed an increase in nonlinear iterations required to converge eigenvalue calculations when using the default norm from Scalable Library for Eigenvalue Problem Computations (SLEPc), $|Bx|_{L_2}$, compared to a postprocessor value based on total neutron production through fission leveraged by MOOSE's legacy eigenvalue executioner. With the merging of PR #25204, the newer SLEPc-based eigenvalue executioner has a superset of the legacy executioner's capabilities, meaning the legacy executioner can be removed, reducing the maintenance burden on both MOOSE and Griffin teams.

5.2 Allow Selective Exclusion of Finite Element Families from P-Refinement

Continued from the prior fiscal year, the Griffin team wanted to be able to perform polynomial (p) refinement in control drum regions in order to remove numerical artifacts. However, in these multiphysics calculations, p-refinement was limited by the fact that Lagrange variables used for `MultiApp` transfers cannot be p-refined beyond the geometric element order, which is generally limited to two. In that vein, we merged PR #25108, which allows users to specify finite element families they do not wish to p-refine. With this feature, Griffin can disable p-refinement of Lagrange transfer variables while allowing p-refinement of the discontinuous basis functions used for the nonlinear variables in its transport calculation, allowing the removal of numerical artifacts in the control drum region.

5.3 Avoiding Unnecessary Expensive Material Property Computation

Griffin has some very expensive cross-section material property evaluations. Some of these properties may not be active depending on the simulation type, and avoiding their unnecessary expensive evaluation is highly desirable. In that vein, between PR #26720 and PR #26696 we added Application Programming Interfaces (APIs) to check whether a given material property is active in a simulation. Using these APIs, developers in applications such as Griffin can optionally evaluate material properties.

5.4 Support for Coordinate Transformations in General Field Transfers

PR #25945 added support for coordinate transformations to the general field transfers. General field transfers are the current workhorse for transferring fields between applications in `MultiApps` setups. Coordinate transformations are mappings between simulations that do not use the same coordinate system. For example, one application may have their frame of reference rotated, translated, scaled, or all of the above. Another common use case is for one application to use Cartesian coordinates, while the other uses cylindrical or spherical coordinates. Coordinate transformations help transfer fields to the matching locations in both frames.

The particular use case at hand was a neutronics simulation in XYZ coordinates, transferring information to and from fuel performance simulations using a two-dimensional cylindrical frame of reference.

5.5 MeshDivisions System for Homogenized Transfers

Heterogeneous models in neutronics consume a lot of computational resources and are generally considered high-fidelity, as they can resolve fine details of the neutron flux solutions, such as plutonium buildup on the rims of fuel pins. For sodium fast reactors, a middle ground has been found by homogenizing each fuel assembly by ring. This is called the ring-heterogeneous model [13, 14], and it uses discrete ordinate transport at a reasonable computational cost.

However, to perform multiphysics simulations, the hexagonal rings have to be mapped to a representative fuel pin in BISON. To perform this mapping, the quantities on the rings, such as the power density or the neutron damage, have to be averaged and transferred to the fuel pin. Similarly, the fuel pin quantities, such as the temperature, have to be mapped to each ring they represent. To do this mapping, we could have created a custom transfer dedicated to this core, which would have been added to BlueCRAB. However, doing so would have created yet another `Transfer` class to maintain. Instead, we introduced the `MeshDivisions` system, which creates numbered divisions of the mesh. These divisions can then be mapped to child applications or other divisions. For the case at hand, a `HexagonalGridDivision` was created.

The numbering of fuel pins needed to follow a convention. A coordination meeting was held

between the points of contact of SAM, Subchannel, Pronghorn, Cardinal, and Griffin to decide on the convention to adopt for the numbering of the hexagonal lattice pins. The Pronghorn-Cardinal convention was selected. Several utilities were moved from Cardinal to MOOSE, to be shared between all MOOSE-based applications. Other applications will migrate over time to this convention, as needed when coupling codes together.

In order to test the system extensively and to jump-start its use by other modelers, the following additional `MeshDivision` objects were created:

- `HexagonalGridDivision` as mentioned previously.
- `CartesianGridDivision` to perform such mappings on square-lattice reactors.
- `CylindricalGridDivision` to handle mappings between cylinders, for example fuel pins.
- `SphericalGridDivision` for numbering layers and sectors of pebbles in pebble-bed reactors.
- `ExtraElementIntegerDivision` to map between mesh regions already numbered using extra element integers. These integers are extensively used in Griffin for material IDs, assembly IDs, and others.
- `NearestPositionsDivision` to use the `Positions` system to create a division of the mesh, based on the nearest-neighbor partition.
- `NestedDivision` to nest division indexings. For example, a hexagonal sodium fast reactor core is composed of a hexagonal lattice of assemblies, themselves composed of a hexagonal lattice of pins. These lattices can be numbered in a nested manner using this object.
- `SubdomainsDivision` to use the mesh subdomains to create the numbered division.

The system was later leveraged to perform reductions (average, min/max, integrals) on the numbered divisions. These reductions are used for transfer purposes, or for postprocessing needs. The `MeshDivisionFunctorReductionVectorPostprocessor` can perform a number of reductions on any functor (variables, functions, functor material properties) following any of the `MeshDivisions` for the binning scheme. For example, Figures 3, 4 and 5 show various divisions of the mesh for performing these reduction operations for various advanced reactor concepts.

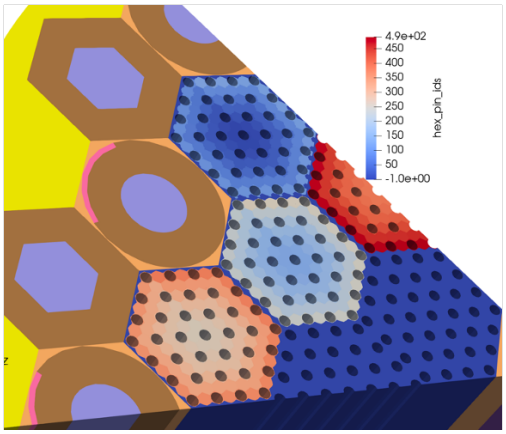


Figure 3: Numbering of pins in selected assemblies in a micro-reactor, allowing to compute pin power rates.

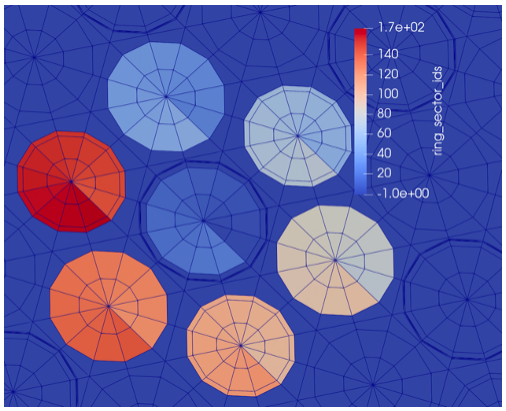


Figure 4: Numbering of rings and sectors in selected fuel pins, to compute local intra-pin powers.

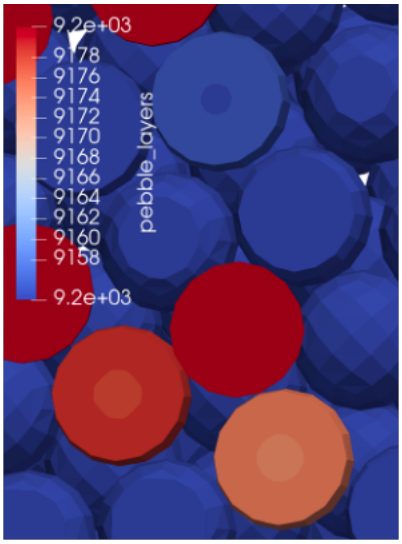


Figure 5: Numbering of spherical shells of individual pebbles in a high temperature gas reactor.

5.6 Ongoing Memory Optimization

PR #25780: Years ago, Griffin developers requested reduction of unnecessary memory consumption via the (potentially optional) removal of the right hand side (RHS) vector used in both implicit and explicit system solves, from the MOOSE auxilliary system and/or nonlinear system(s). This year, the issue was raised again, with performance numbers from Griffin quantifying the potential benefit to their RAM requirements, and we attempted to achieve as much of this memory usage reduction as possible. In PR #25780, we changed the MOOSE `AuxilliarySystem` class to internally create a plain `libMesh::System` (designed to store finite element and/or finite volume data) rather than the previously created `libMesh::ExplicitSystem` (designed to additionally store a residual to enable time integration of that solution data), eliminating the `RHS` vector which the `ExplicitSystem` was automatically allocating. Though this was done with Griffin in mind, and has the greatest relative impact for applications with memory usage characteristics resembling Griffin's, it is an optimization for all MOOSE-based applications.

5.7 Eigenvalue Calculations with Constraints

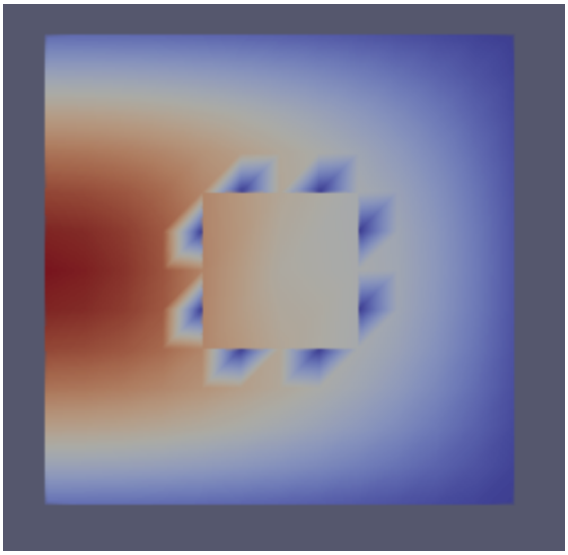
PR #27276 enabled eigenvalue computations for problems with constraints such as hanging nodes due to mesh adaptivity, periodic boundary conditions, or libMesh Dirichlet boundary conditions. This work was spurred by the High Temperature Gas Reactor (HTGR) application driver. The improvements of PR #27276 are illustrated in Figure 6.

5.8 Restore Auxiliary System During Fixed Point Iterations

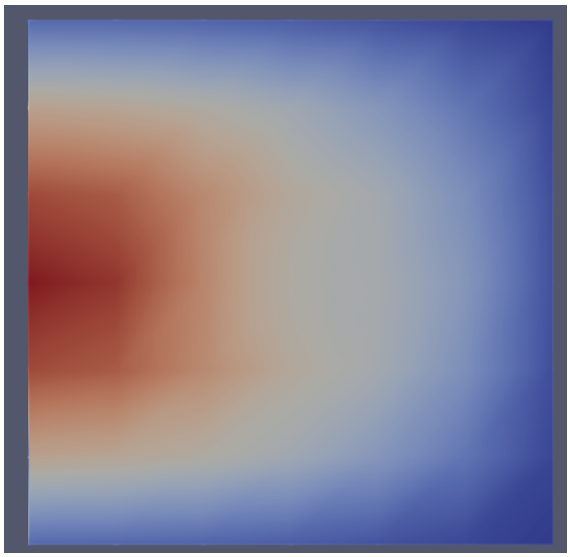
As a Griffin team member outlined in issue #19078 the auxiliary system was not being restored during fixed point iterations, leading to unnecessarily long times to reach steady-state. This is rectified in PR #28703.

6. BISON SUPPORT

Please refer to [15] for more information on the BISON code.



(a) Result of eigenvalue calculation with hanging nodes prior to PR #27276. The spurious modes due to inclusion of the constraints in the eigenvalue calculation are evident.



(b) Result of eigenvalue calculation with hanging nodes after PR #27276. The spurious modes have been removed.

Figure 6: Results of an eigenvalue solve for a Griffin neutronics problem

6.1 Support for Auxiliary Kernel Evaluation of Lower-Dimensional Variables

For some time, MOOSE has supported the calculation of nonlinear lower-dimensional variables through mortar methods. PR #26793 and PR #26625 added support for evaluating lower-dimensional variables using the auxiliary system. These pull requests enable populating these lower-dimensional variables using boundary restricted auxiliary kernels. Use of the boundary restricted auxiliary kernel as opposed to an auxiliary kernel block restricted to the lower-dimensional subdomain enables pulling in face evaluations of higher-dimensional variables and material properties, as well as coincident lower-dimensional variables. PR # 26625 also added significant ghost element robustness improvements for parallel and distributed mesh computations.

7. CARDINAL SUPPORT

Please refer to [16] for more information on the Cardinal code.

7.1 Silencing Transfer Diagnostics Warning

Cardinal simulations often transfer solution fields between NekRS and MOOSE. The solid temperature in MOOSE is passed from the MOOSE mesh nodes to the NekRS mesh nodes. However, the heat flux is passed from the MOOSE mesh surface cell centers to the NekRS mesh nodes. This is because heat fluxes are computed on element faces in MOOSE. Because multiple surface cell centers can be equidistant from a node, the automated diagnostics of the general field transfers in MOOSE report numerous warnings. These warnings can now be silenced to manage console output in Cardinal simulations.

8. SAM SUPPORT

Please refer to [17] for more information on the SAM code.

8.1 Consistent Handling of Optional Input Parameters

It was noted by a SAM team member in issue #24455 that when retrieving a non-required `std::vector` parameter without a default value that MOOSE would not error out as it does for other parameter types if the parameter was not set in the input file. PR #25245 removed this inconsistency, making the behavior for `std::vector` also an error.

9. CONCLUSION

This report highlighted various improvements made to the MOOSE framework in direct support of NEAMS applications. Some of the highlights include enabling selective polynomial basis refinement, building a scalable preconditioner for saddle-point problems, introducing the `MeshDivisions` system to create matching transfer regions, and the `Convergence` system to allow for flexible determination of the convergence of various types of solves. Due to the frequency and number of improvements, technical area support remains a very popular item among application developers in the NEAMS program.

REFERENCES

[1] A. D. Lindsay, D. R. Gaston, C. J. Permann, J. M. Miller, D. Andrš, A. E. Slaughter, F. Kong, J. Hansel, R. W. Carlsen, C. Icenhour, L. Harbour, G. L. Giudicelli, R. H. Stogner, P. German, J. Badger, S. Biswas, L. Chapuis, C. Green, J. Hales, T. Hu, W. Jiang, Y. S. Jung, C. Matthews, Y. Miao, A. Novak, J. W. Peterson, Z. M. Prince, A. Rovinelli, S. Schunert, D. Schwen, B. W. Spencer, S. Veeraraghavan, A. Recuero, D. Yushu, Y. Wang, A. Wilkins, and C. Wong, "2.0 - MOOSE: Enabling massively parallel multiphysics simulation," *SoftwareX*, vol. 20, p. 101202, 2022.

[2] R. L. Williamson, J. D. Hales, S. R. Novascone, G. Pastore, K. A. Gamble, B. W. Spencer, W. Jiang, S. A. Pitts, A. Casagranda, D. Schwen, A. X. Zabriskie, A. Toptan, R. Gardner, C. Matthews, W. Liu, and H. Chen, "Bison: A flexible code for advanced simulation of the performance of multiple nuclear fuel forms," *Nuclear Technology*, vol. 207, no. 7, pp. 954–980, 2021.

[3] M. DeHart, F. N. Gleicher, V. Laboure, J. Ortensi, Z. Prince, S. Schunert, and Y. Wang, "Griffin user manual," Tech. Rep. INL/EXT-19-54247, Idaho National Laboratory, 2020.

[4] MOOSE Team, "Moose github page." `https://github.com/idaholab/moose`, 2014.

[5] S. Schunert, M. E. Tano Retamales, and M. K. Mohammad Jaradat, "Overlapping domain coupling of multidimensional and system codes in neams-pronghorn and sam," tech. rep., Idaho National Laboratory (INL), Idaho Falls, ID (United States), 2023.

[6] S. Schunert, G. Giudicelli, A. Lindsay, P. Balestra, S. Harper, R. Freile, M. Tano, and J. Ragusa, "Deployment of the finite volume method in pronghorn for gas- and salt-cooled pebble-bed reactors," External report INL/EXT-21-63189, Idaho National Laboratory, 2021.

[7] A. Lindsay, G. Giudicelli, P. German, J. Peterson, Y. Wang, R. Freile, D. Andrs, P. Balestra, M. Tano, R. Hu, *et al.*, "Moose navier–stokes module," *SoftwareX*, vol. 23, p. 101503, 2023.

[8] H. C. Elman, "Preconditioning for the steady-state navier–stokes equations with low viscosity," *SIAM Journal on Scientific Computing*, vol. 20, no. 4, pp. 1299–1316, 1999.

[9] H. Elman, V. E. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro, "Block preconditioners based on approximate commutators," *SIAM Journal on Scientific Computing*, vol. 27, no. 5, pp. 1651–1668, 2006.

[10] IAPWS, "Iapws: Guideline on the fast calculation of steam and water properties with the spline-based table look-up method." 2015.

[11] M. Kunick, R. A. Berry, R. C. Martineau, H.-J. Kretzschmar, and U. Gampe, "Application of the new iapws guideline on the fast and accurate calculation of steam and water properties with the spline-based table look-up method (sbtl) in relap-7," *Kerntechnik*, vol. 82, no. 3, pp. 264–279, 2017.

[12] Y. Wang, Z. M. Prince, O. W. Calvin, H. Park, N. Choi, Y. S. Jung, S. Schunert, S. Kumar, J. Hanophy, V. Laboure, *et al.*, "Griffin: A moose-based reactor physics application for multiphysics simulation of advanced nuclear reactors," *Available at SSRN 4844473*.

[13] S. Terlizzi, N. Choi, J. R. Harter, I. Trivedi, T. Mui, and J. Ortensi, "Sodium-cooled fast reactor reference plant model," 3 2024.

[14] N. Choi, S. Terlizzi, Y. Wang, H. Park, C.-h. Lee, and J. Ortensi, "Investigation of ring-heterogeneous geometry approximation for efficient and practical sfr analysis," 4 2024.

[15] R. L. Williamson, J. D. Hales, S. R. Novascone, G. Pastore, K. A. Gamble, B. W. Spencer, W. Jiang, S. A. Pitts, A. Casagranda, D. Schwen, A. X. Zabriskie, A. Toptan, R. Gardner, C. Matthews, W. Liu, and H. Chen, "BISON: A flexible code for advanced simulation of the performance of multiple nuclear fuel forms," *Nuclear Technology*, vol. 207, no. 7, pp. 954–980, 2021.

[16] A. Novak, D. Andrs, P. Shriwise, J. Fang, H. Yuan, D. Shaver, E. Merzari, P. Romano, and R. Martineau, "Coupled Monte Carlo and Thermal-Fluid Modeling of High Temperature Gas Reactors Using Cardinal," *Annals of Nuclear Energy*, vol. 177, p. 109310, 2022.

[17] R. Hu, L. Zou, G. Hu, D. Nunez, T. Mui, and T. Fei, "SAM Theory Manual," Tech. Rep. ANL/NSE-17/4 Rev. 1, Argonne National Laborary, Argonne, IL (United States), 2021.