INL/RPT-23-74787

NEAMS Technical Area Support in MOOSE

September 2023

Nuclear Energy Advanced Modeling and Simulation M3 Milestone September 2023

Alexander Lindsay[1], Guillaume Giudicelli[1], Mengnan Li[1], Logan Harbour[1], Derek Gaston[1], and Cody Permann[1]

[1]*COMPUTATIONAL FRAMEWORKS*

INL/RPT-23-74787

NEAMS Technical Area Support in MOOSE

Nuclear Energy Advanced Modeling and Simulation M3 Milestone
September 2023

Alexander Lindsay[1], Guillaume Giudicelli[1], Mengnan Li[1], Logan Harbour[1], Derek Gaston[1], and Cody Permann[1]

[1]COMPUTATIONAL FRAMEWORKS

September 2023

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
AL	Augmented Lagrange
ANL	Argonne National Laboratory
API	Application Programming Interface
MOOSE	Multiphysics Object-Oriented Simulation Environment
NEAMS	Nuclear Energy Advanced Modeling and Simulation
SFR	Sodium Fast Reactor
TA	Technical Area
TH	Thermal Hydraulics

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 transcribed in this report was conducted in direct support of the simulation tools and has already been deployed. The capabilities outlined in this report include addition of Times and Positions systems, redesign of mechanical contact constraints to enable the augmented Lagrange algorithm, overhaul of the restart system, and incorporation of p-refinement into MOOSE.

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 LGPL 2.1 license, 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 New functor interfaces

Similar to traditional material properties, the functor system initially required that the automatic differentiation "status" match between functor producers and consumers, e.g., if a functor was declared as an Automatic Differentiation (AD) type, then all consumers of the functor had to retrieve it as an AD type. However, this is unnecessarily restrictive. For instance, an auxiliary kernel will never need automatic differentiation derivative information, so conceptually, they should always be able to retrieve functors as non-AD types. This restriction was removed through introduction of two new interfaces: `NonADFunctorInterface` and `ADFunctorInterface`. Objects that inherit `NonADFunctorInterface` (like auxiliary kernels) may always retrieve functors as non-AD. Behind the scenes in this instance, functors which are declared as AD types will have non-AD copies created on the `SubProblem` which evaluates the AD version and then calls the `MetaPhysicL::raw_value` function to strip the derivatives. Conversely, objects that inherit the `ADFunctorInterface`, like residual and Jacobian objects, will "promote" non-AD declared functors to corresponding AD types when requested by the residual/Jacobian object. The only error case is if a `ADFunctorInterface` object retrieves a functor as a non-glsad type, but the functor has, in fact, been declared as an AD type. These new interfaces have allowed significant code simplification, e.g., removal of templating of the `Function` class and a lot of `is_ad` templating in classes like auxiliary kernels and postprocessors.

3.2 Ability to rename parameters and coupled variables

Oftentimes, a derived class may want to rename a general base class parameter to something more meaningful and specific to the derived classes purpose. In PR #23260 we added this capability. Base class parameters and coupled variables may now be renamed and have their documentation strings re-specified. This PR also introduced new parameter deprecation Application Programming Interfaces (APIs) that are appropriate when one parameter name is being deprecated in favor of a new name (at the same class level). The new APIs leverage maps from old/deprecated names to new names. These maps are also used in parameter retrievers, such that developers can check only the "blessed" name when retrieving a parameter. In the past developers had to check all possible aliases of a parameter name (e.g., the blessed name plus deprecated names) when determining whether a user specified a parameter. This improvement also has the beneficial side-effect that blessed parameter names may be declared as required even when deprecated names are supported; in the past, the blessed name had to be non-required when deprecated aliases were possible.

3.3 Ensuring the mesh is properly prepared

During the previous fiscal year, a user reported incorrect results when performing a heat conduction calculation with Pronghorn's finite volume implementation. The issue turned out to be missing neighbor links in the mesh which prevented proper calculation of the conduction fluxes at the corresponding faces. To prevent this kind of error from ever happening again, a final mesh `prepare_for_use` call was inserted that executes at the end of the mesh generation phase if the mesh is marked as unprepared. In addition a debugging check at the end of *every* mesh generator's execution was added that verifies if a mesh is truly prepared if it as marked as such.

3.4 Minor enhancements and bug fixes

- PR #23588 ensures that `VectorPostprocessorFunction` evaluations are parallel consistent.
- PR #23665 made global sparse indexing the only possible automatic differentiation configuration in MOOSE. While potentially faster for continuous finite elements, the local nonsparse configuration that was removed was much less flexible; it could not be used for finite volume calculations, which can involve stencils of up to three element neighbor layers. Removing the nonsparse configuration allowed removal of over 2,000 lines of code in MOOSE, significantly reducing maintenance burden for framework developers.
- PR #23752: Simulations with numerous postprocessors or scalar variables, such as simulations using the SAM application, could overwhelm the console and make it essentially impossible to read tables of numbers. By changing the ordering of the default parameters for console table output, we ensured that the console output remained legible.
- PR #23735: Subdomains with different IDs are allowed to have the same subdomain name. However, this is not intuitive and is confusing to end-users. A subdomain name duplication check function was added to `MooseMesh`. It returns an error message if the name duplication is found for subdomains with different IDs.

4. PRONGHORN SUPPORT

Please refer to [5] and [6] for more information on the Pronghorn code.

4.1 Evaluation of functors at previous nonlinear iterations

Some physics are very nonlinear, and Newton's method is not guaranteed to be globally convergent. One method to reduce nonlinearity is to replace evaluation with the current solution state with the previous nonlinear iteration solution state. MOOSE finite element simulations have long had the ability to use solution values from the previous nonlinear iteration. However, MOOSE's finite volume implementation, which primarily uses functors, has not. In PR #23893 we generalized the "temporal" argument of functors to be able to handle previous solution states, including old time steps and old nonlinear iterations. This should allow more robust solution of highly nonlinear physics, such as turbulent flows—the original motivating application for this work.

4.2 Performance enhancements for finite volume simulations

A noticeable slowdown in finite volume simulations was observed after merging a new porous flow treatment capability for NEAMS Thermal Hydraulics (TH). We were able to make up for the slowdowns and achieve greater speedup through three different optimizations in PR #23233:

- caching lookup of Dirichlet boundary conditions,
- not performing pre-init calculations (which are required for finite elements, but not finite volumes), and
- more caching of degree of freedom index lookups.

4.3 Integrated constraint generalization

Several generalizations to constraints for finite volume simulations were implemented in PR #23769:

- Support was added for imposing constraints integrated over boundaries instead of blocks.
- The target constraint value was changed to a `PostprocessorValue` from an ordinary `Real`. This allowed domain overlapping treatment between Pronghorn and SAM in which the

target pressure value on the boundary for Pronghorn could be updated over time based on the system plant calculations in SAM.

Furthermore, the integrated pressure constraint condition was verified by ensuring its imposition on an outlet boundary did not disrupt a fully developed channel flow profile and that the resulting pressure field matched the desired constraint value as shown in Figure 1.

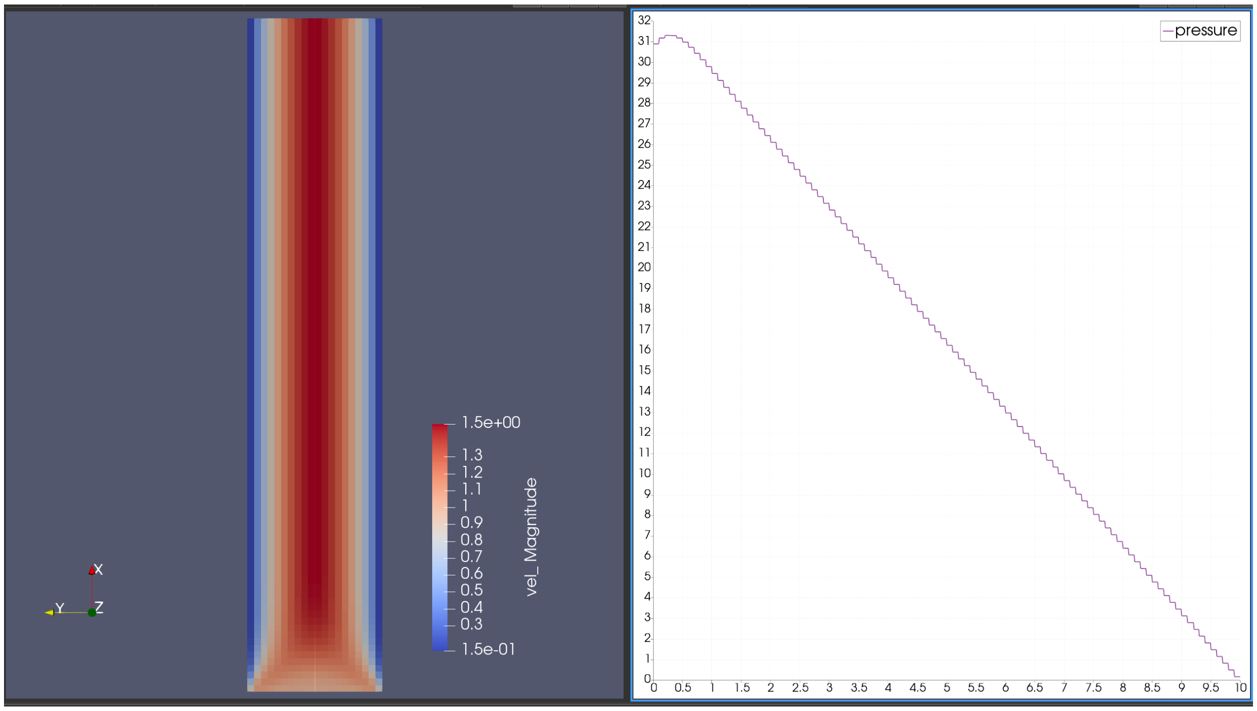

Figure 1: Verification of an integrated pressure constraint. The average pressure is constrained to be zero on the top boundary. The fully developed flow profile shown in the left is not perturbed by the integrated constraint, while the pressure goes to zero at the outlet as expected.

4.4 Functor thermal solid properties

The finite volume implementation in MOOSE mostly, and increasingly, relies on the `Functor` system to provide material properties. The new Solid Properties module did not provide the necessary functor material to enable the use of the its solid properties objects. This was added in PR #24351 to support a study using Pronghorn and solid properties for the porous medium.

4.5 Controllable finite volume boundary conditions

PR #24446 added support for controlling (e.g. enabling/disabling) finite volume Dirichlet boundary conditions. Previously, the Dirichlet conditions were determined at construction time. This work was conducted in support of Pronghorn-SAM domain overlapping coupling.

4.6 Minor enhancements and bug fixes

Several minor items necessitated timely development to enable the continuation of NEAMS-funded Pronghorn research:

- PR #24617 ensured that derivative calculations will not occur when calling residual evaluations from explicit time integrators. This change sped up an explicit time integration simulation with Pronghorn by a factor of two.
- In PR #24521 finite-element/finite-volume mixed variable sampling was enabled in order to be able to gather information (output at specific locations) from coupled simulations.
- PR #24382 added the ParsedVectorAux auxiliary kernel to be able to output anisotropic material properties that are common in reactor porous flow simulation. Friction coefficients are very anisotropic when representing flow channels, such as those in a Sodium-cooled Fast Reactor (SFR) fuel assembly, as a porous medium.
- PR #24808 allows Schur complement field splits, which are directly applicable to incompressible Navier-Stokes, to be run with the full inverse of the A block, which is a relatively new debugging option introduced in PETSc. Using the full inverse for A yields an exact representation of the Schur complement, which is dense and consequently both memory and computationally intensive. So, the full inverse option is not meant to be used in production, but it allows for checking whether a Schur complement method can effectively precondition a given problem.

5. GRIFFIN SUPPORT

Please refer to [7] for more information on Griffin.

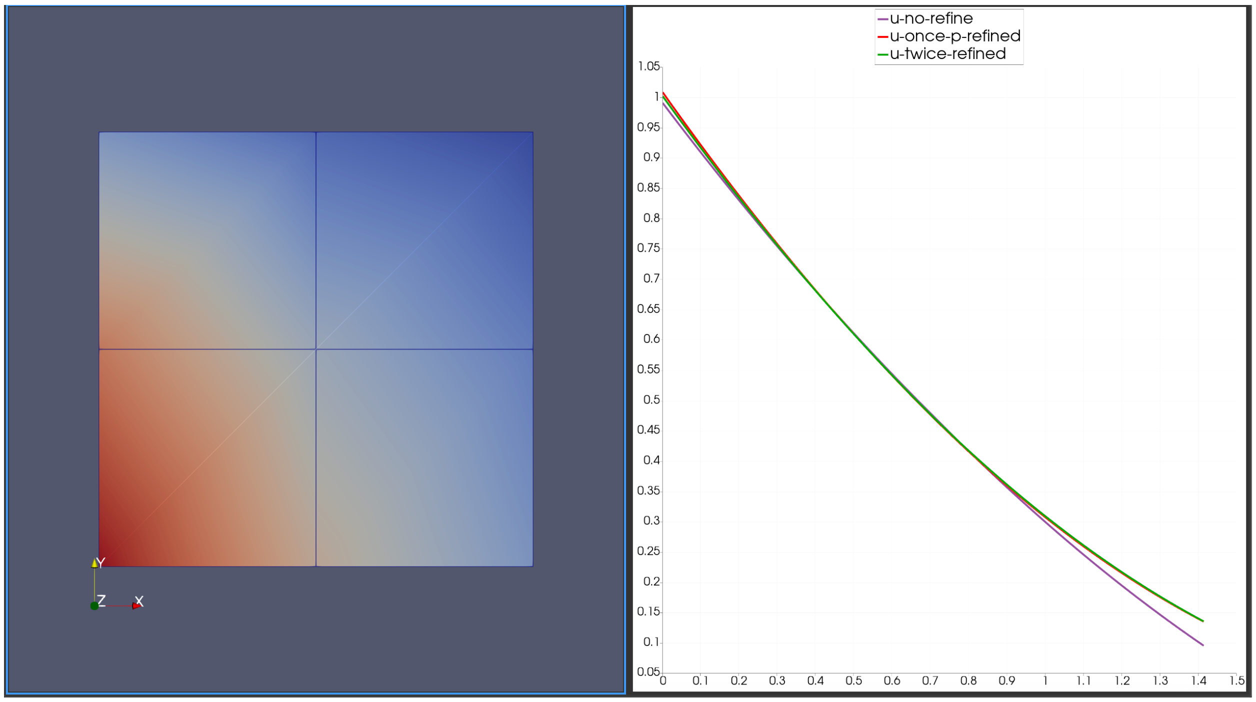

Figure 2: Illustration of p-refinement for a two-dimensional diffusion problem with a custom source. On the left we show the solution for the finest p-level simulation on the 2x2 element grid. On the right, the solution traced on a line from the bottom-left to top-right corners is plotted for zero, one, and two levels of p-refinement. From zero to one levels of refinement, there is a significant visual improvement in the solution, while from one to two levels of refinement the visual change decreases.

5.1 Support for polynomial basis refinement

Polynomial basis refinement, or p-refinement, enables exponential accuracy convergence rates in regions with a smooth solution. P-refinement has long been a capability in libMesh, but it was not available in MOOSE until the Griffin team requested its incorporation this fiscal year for simulating control drum rotations. In Figure 2 we demonstrate the application of p-refinement to a two dimensional diffusion problem. Initial incorporation of p-refinement in MOOSE was performed in PR #24180. Follow-on work includes selectively enabling/disabling p-refinement for specific variable groups. For instance, when running Griffin with a discontinuous Galerkin discretization, the user may want to allow p-refinement for the nonlinear discontinuous Galerkin variables while disabling p-refinement for auxiliary Lagrange variables that may be transferred via `MultiApps`.

5.2 Support for restarting an eigenvalue into a transient

Reactor analysis often requires the transition from a steady state analysis into a transient analysis. In particular, this involves running a steady state problem with the `Eigenvalue` executioner in MOOSE to get the flux shape and eigenvalue. Then, that state is used for the initial condition for a problem that uses the `Transient` executioner.

MOOSE has a restart capability that enables the loading of previous solutions and states into a problem with different parameters and objects. Unfortunately, the specific case of transitioning from a steady state eigenvalue problem to a transient problem was not supported with this system. The system was lacking a significant level of flexibility. From this, a significant rework of the restart system in MOOSE was accomplished. It now supports the system- and executioner-agnostic restart of solution and non-solution vectors. In addition, it supports the late load of restartable information. This work was completed across PR #24762, PR #24763, PR #24510, and PR #25415.

5.3 Block restriction of copy transfers

Copy transfers are the simplest form of field transfers. They copy one or more fields from one application to another, and they are represented in exactly the same form in both applications. Both applications use the same mesh, and the fields are of the same finite element family and of the same finite element order. To copy these fields/variables, we can simply pass the degree of freedom values and copy them at the same indexes in the target variable.

This was previously only possible over the entire mesh. A user in the Griffin team requested this be possible on a per-subdomain basis for the problem shown in Fig. 3. This was implemented in PR #22277 by splitting off the copy transfer into a new derived class. In Griffin, another transfer was inheriting the copy transfer class and could not support block restriction.

5.4 Minor enhancements and bug fixes

Several minor items necessitated timely development to enable the continuation of NEAMS-funded Griffin research:

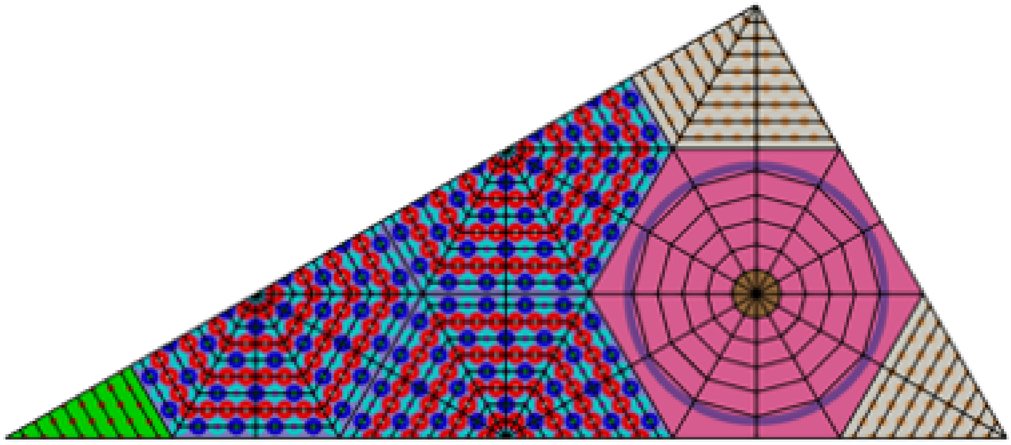

Figure 3: Coarse mesh for which the microreactor case copy transfer block restriction was required. This mesh is shared between two applications.

- PR #23407 allows the hybrid finite element method to be used with a distributed mesh. More specifically, remote element deletion is delayed until just before the equation systems are initialized, so that the memory loads of a replicated mesh and solution vectors are not incurred simultaneously.

- Users of the reactor module, in particular from the Griffin team, have reported extraordinary verbosity from the stitching operations, especially when generating lattices. Creating lattices often involves stitching hundreds of pin cell meshes. Because a stitching operation can be performed on only partially overlapping nodesets, the default for this operation is to output five lines on the console describing the two boundaries to stitch and the output of the stitching. For lattice stitching operation however, the stitching is usually entirely successful so this logging is unnecessary. It was turned off at their request in several mesh generators in PR #23518.

- PR #25033 ensures that the displaced problem will correctly report the accessibility status of tagged vectors and matrices. Previously, as reported by the Griffin team when running with a displaced mesh, the displaced mesh could report safe access statuses out of sync with the `FEProblem`, resulting in bad access to tagged vectors/matrices.

- PR #25039 removes the creation of an additional vector when running a transient. This extra vector was used to compute the difference in solutions between timesteps for contributing to convergence criteria. This difference is now computed in place and does not require an extra allocation. This contributes to decreasing the overall memory footprint of Griffin.

6. BISON SUPPORT

Please refer to [8] for more information on the BISON code.

6.1 Accurate transfers from displaced meshes

As outlined in issue #22534, transfers from displaced meshes could be problematic. An illustration of a bad transfer is shown in Figure 4, where the mesh's point locator was incorrectly regarded as valid after displacing the mesh, meaning that transfer points were mapping to the wrong elements for transfer. After properly invalidating the point locator after mesh displacement, the transfer results were fixed as shown in Figure 5.

6.2 Enabling Augmented Lagrange mortar contact

MOOSE has supported mortar contact using Lagrange multipliers for several years. However, the Lagrange multiplier approach has notable drawbacks. It adds additional degrees of freedom to the nonlinear system, which changes the condition number and eigenvalue structure of the linearized operator, preventing application of iterative solvers. An Augmented Lagrange (AL) approach avoids these issues. However, in order to effectively implement AL, the structure of mortar constraint application had to be re-designed. PR #23591 refactored much of the code from mortar constraint classes into user objects, making extensibility of mortar methods to alternative implementations of mechanical contact (such as penalty and AL) feasible. Since the merging of #23591, the BISON team has conducted meaningful work using the AL method, demonstrating use of iterative solvers with mortar mechanical contact. Verification of the AL method enabled by #23591 is shown in Figure 6 and Figure 7.

6.3 General handling of output time in a simulation

A member of the BISON team needed their simulation to output at certain simulation times. This usually leverages the `sync_times` parameter of the *Outputs* system. This parameter triggers

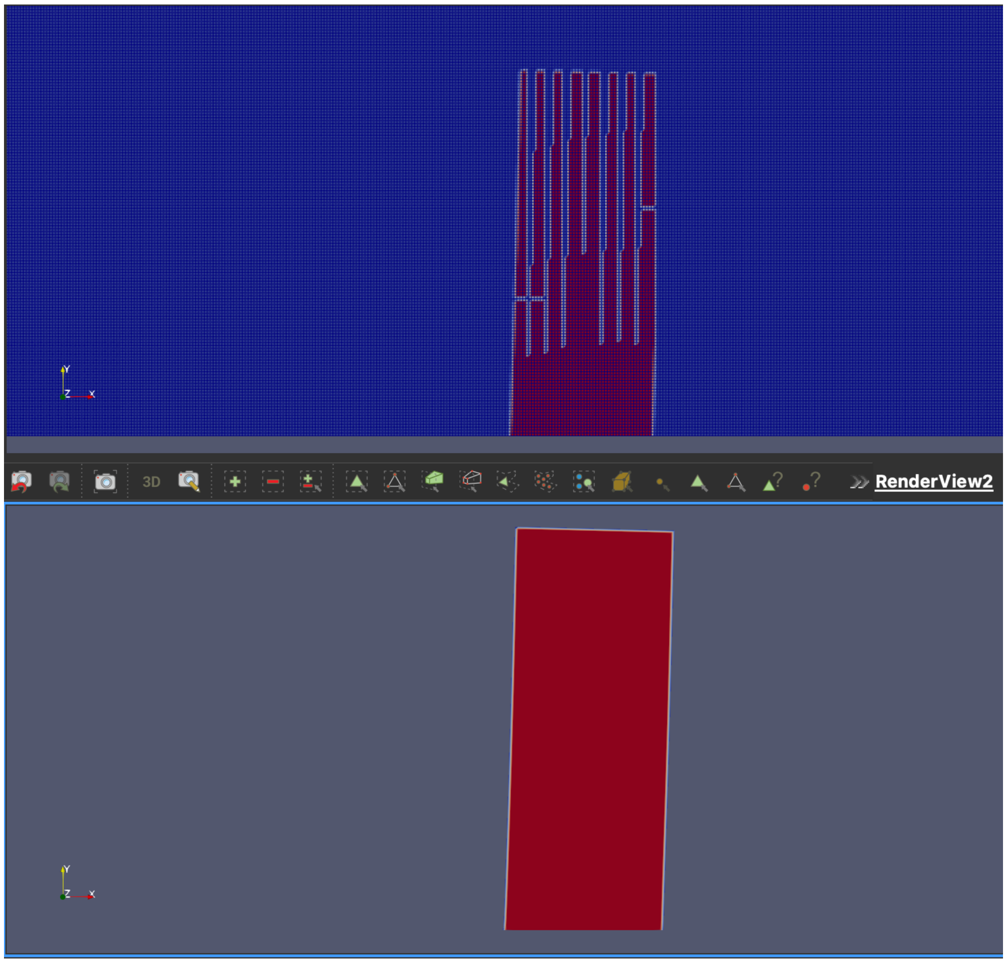

Figure 4: Illustration of a poor transfer from a displaced mesh. The top of the figure shows the transferred field on the “to” mesh, which is clearly different from the field on the “from” mesh.

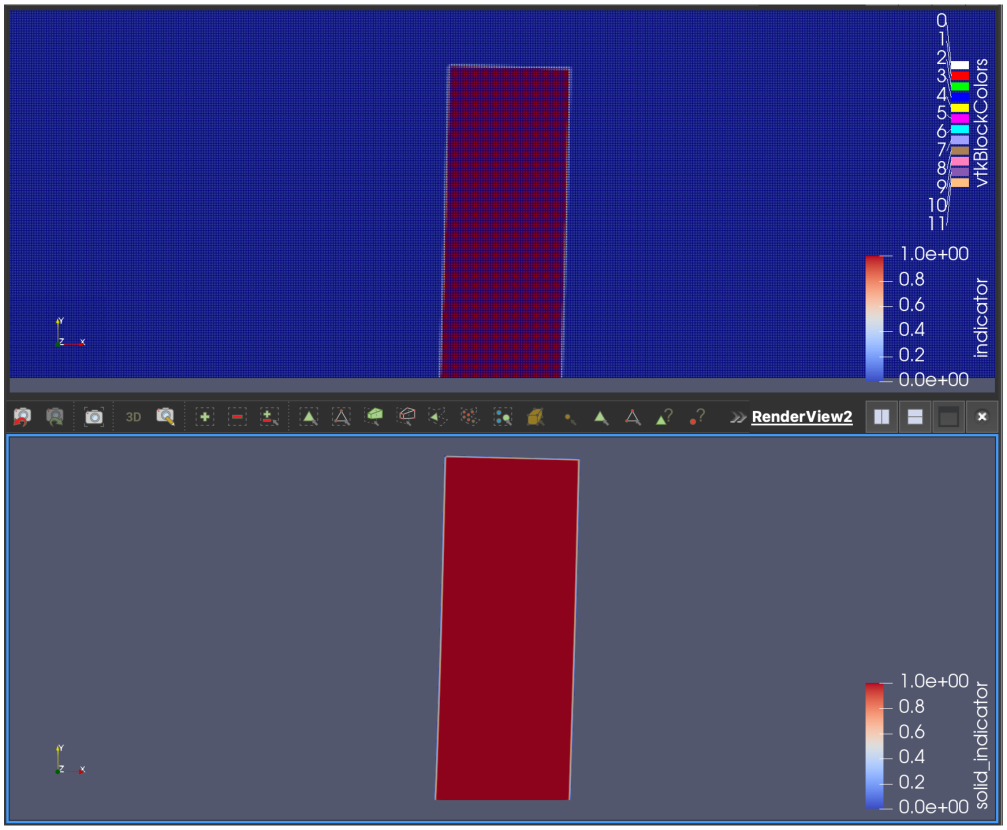

Figure 5: Good transfer results from a displaced mesh. The top of the figure shows the transferred field on the “to” mesh, while the bottom of the figure shows the field on the “from” mesh.

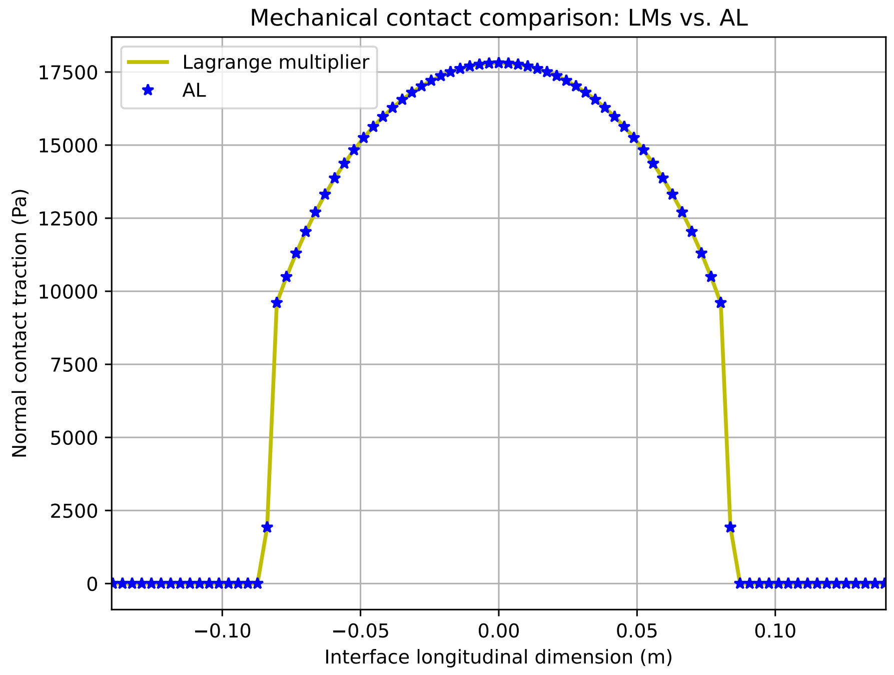

Figure 6: Verification of AL method results for the normal component of traction.

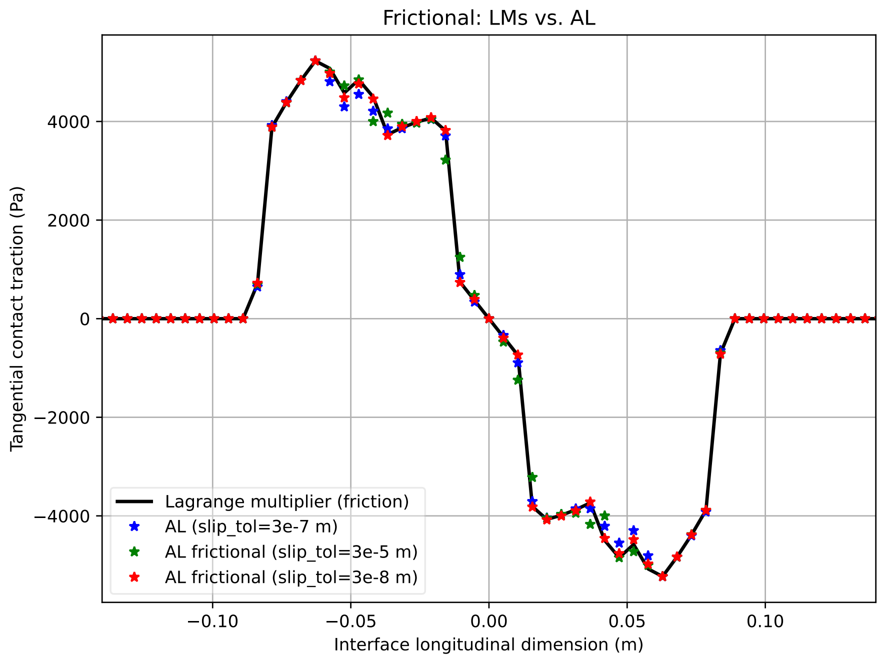

Figure 7: Verification of AL method results for the tangential component of traction.

synchronization times in the simulation. At these times, the solver obtains a solution, and it is then output. This parameter allowed for user input of the list of times directly in the input file. Unfortunately, the list of times to output was very long and realistically needed to be provided in a separate text file.

While allowing for a temporary workaround to be merged in MOOSE, we developed the *Times* system in MOOSE in PR #21243 to respond to this need as well as provide tremendous new capabilities. This system hosts a vector of times, which can be loaded from:

- a list in the input file,
- a list in a text file,
- the times used in an Exodus file,
- a regular time interval [start, end],
- the dynamic evaluation of a Functor (variable, postprocessor, function, functor material property),
- a Reporter object (dynamically), and
- the simulation times (dynamically).

Once loaded in a Times object, we can then pull in these times

- in a time stepper, to select the next time step. This can allow us to easily create custom, dynamic time steps that, for example, can be communicated as reporters between applications. The workflow to achieve this simulation scheduling behavior is both simplified (no need to code) and centralized (objects can pull in the same times),
- or in an output object, to output at the desired times. This meets the original need as the output times can now be loaded in the desired way.

6.4 Increased capability in the tagging interface

The BISON team recently added a capability for building reference residual vectors using the summation of the absolute values of local residuals. However, as initially implemented, the capability was a little brittle and only worked for a subset of residual objects. In PR #24206 this capability was generalized to work for all residual objects, requiring that all residual and Jacobian contributions from a residual object be processed through the `TaggingInterface`. By forcing residuals and Jacobians to funnel through the `TaggingInterface`, developers can be sure that operations such as the application of an absolute value operator can be uniformly applied. This capability has been successfully demonstrated in simulations with both hand-coded Jacobian and AD objects, the latter of which did not previously support the absolute value reference residual capability.

6.5 Minor enhancements and bug fixes

Two minor items necessitated timely development to enable the continuation of NEAMS-funded BISON research:

- PR #24348 established that the reference points from the neighboring element on the undisplaced mesh will always be used for reinitializing the finite element objects on the neighboring element in the displaced mesh. This represents both a performance optimization and a fix for cohesive zone mechanics, in which the the displaced neighbor reference points cannot be determined by inverting the physical location of the displaced *element* reference points because the faces are not coincident.
- PR #25043 added a helpful error message for users attempting to run refinement mesh generators before adding lower dimensional blocks.

7. Cardinal support

Please refer to [9] for more information on the Cardinal code.

7.1 Restricting variable conversions in transfers

NekRS supports the specification of boundary heat fluxes on nodes. The MOOSE heat conduction module supports the computation of boundary heat fluxes on element faces, and the value is effectively stored on the centroid of the boundary element. This is problematic when matching the MOOSE mesh and the NekRS mesh and passing the values. The meshes do not overlap; the MOOSE mesh represents the solid, the NekRS mesh represents the fluid, and the nodes do not match between both meshes. It is, therefore, not possible to evaluate the shape function of the MOOSE variable on the node. To transfer the variable, Cardinal has long resorted to a nearest-node transfer, with a renormalization process to conserve heat flux.

The nearest-node transfer was poorly named and was actually resorting to centroid values for constant monomial variables. This was causing issues with parallel reproducibility as several nodes can be equidistant to a single face centroid (for example, with a quadrilateral face) and, similarly, several face centroids can be equidistant to a node in a regular mesh.

Several actions were taken in PR #24228:

- The nearest-node transfer was renamed to the nearest-location transfer to convey this important distinction for constant variables.
- The general field of the nearest-location transfer examines the source and target variable and now produces a warning to warn the user of the likely equidistance between several sources and a target point. It prompts the user to resort to a Projection auxiliary kernel, detailed in the next subsection.
- Several fixes were implemented to properly evaluate various node-continuous/discontinuous finite element variable families on the nodes/centroids of the element respectively in that transfer.

7.2 Projecting variables before transfers

A workaround for the transfers issue presented earlier is to project the source variable of a transfer to a finite element type that is more compatible with the target variable. This was somewhat possible in MOOSE using the `SelfAux` auxiliary kernel. This kernel would evaluate a source variable at all the quadrature points then effectively perform a least-square projection onto the variable it was setting. Beyond the poor name, which was changed to `ProjectionAux`, the capability was extended in PR #24237 to support projections:

- from variables defined on elemental degrees of freedom to variables defined on nodal degrees of freedom,

- including variables discontinuous at nodes, with a element-volume weighted averaging, and
- from variables defined on nodal degrees of freedom to variables defined on elemental degrees of freedom.

Additionally, certain variable types in libMesh that require gradient degrees of freedom were not supported and were still allowed by the input syntax. They are now explicitly unsupported and a user may not specify them in the input file when using the auxiliary kernel system.

7.3 General spatial restrictions transfers

Users of Cardinal at Argonne National Laboratory (ANL) relying on nearest-position based transfers to pass heat fluxes between the MOOSE heat conduction module and NekRS have reported that heat fluxes for a Sodium Fast Reactor (SFR) assembly duct were being passed to the nearest cladding surface and vice-versa. This is an unfortunate consequence of using nearest-position transfers without any additional restriction. On the cladding surface, the transfer looks for the nearest node source, whether it be on the duct or the clad. Similarly, if the meshes are severely misaligned axially between neighboring fuel pins and their cladding, it is easy to imagine transferred values to originate from the wrong cladding regions because its nodes are nearest in the region of interest.

In order to combat that, the *Positions* system was created in PR #24071. This new system keeps track of vectors of points, including multi-dimensional vectors up to 4D, that can be specified in numerous ways in the user input, such as:

- a list of points in the input file,
- a list of points in a text file,
- a vector of points provided by a Reporter object, which enables transferring positions between applications,

- the dynamic evaluation of 3 Functors for the X, Y, and Z coordinates, which enables the dynamic generation and update of positions,
- from the current positions or mesh centroids of multiapps, which enables tracking the displacement of multiapps through time,
- from element centroids in a mesh, which reproduces the functionality of the `CentroidMultiApp` and also enabled the creation of element-centroid based non-transient multiapps (a need of the Griffin team), and
- from centroids of groups of elements in a mesh. This was created to be able to obtain positions from a mesh with assemblies or fuel pins specified using extra element integer ids or subdomains, as in the reactor module.

Using the *Positions* system, it is now possible to:

- anchor transfers for a given target location to the nearest position. Only source values that are nearest to this target location are considered. This avoids contamination of values between fuel pins, if the positions used are at the center of the respective pins.
- create multiapps at these positions. Not only does this introduce numerous new ways to define positions, but also the positions can change during the simulation in a manner that is much more flexible and general than the previous "move" parameters.

7.4 Specific execution schedules for avoiding fixed point multiapp iterations

Cardinal simulations typically execute on three very different time schedules. The OpenMC simulation is generally steady state eigenvalue calculations. The heat conduction simulation is usually using seconds or tenths of seconds time steps. The computational fluid dynamics model is using an explicit time integration scheme and uses much smaller time steps, depending on the problem.

This dissimilarity in time steps makes the current framework for fixed point iterations very expensive to run. Indeed, MOOSE currently iterates over all multiapps on every fixed point iteration to convergence by default. Previously, there was not an ability for the user to decide to iterate with one application and not with another. In PR #23065 we introduced the `MULTIAPP_FIXED_POINT_BEGIN` and the `MULTIAPP_FIXED_POINT_END` execution schedules for that purpose.

An object executed on the former is only executed once before the fixed point iteration of applications. If that object is a multiapp, for example, then it will use information at the beginning of the time step to compute its solution and transfer it at the end of its own time step(s). This corresponds to an implicit-explicit time integration scheme. Selection of the appropriate time integration scheme, especially in the context of multiapps, remains the prerogative of the user and/or application developer.

7.5 Minor enhancements and bug fixes

One minor item necessitated timely development to enable the continuation of NEAMS-funded Cardinal research:

- PR #23091 demonstrated how to avoid an issue reported in PR #17534 in which an `AuxKernel` evaluation was required in order to get the correct results in a `Postprocessor`. The pull request shows that it is not enough to update the parallel solution; a follow-on `System::update()` step is required in order to propagate the changes in the parallel solution vector to the ghosted vector which is used by consumer objects such as `Postprocessor`.

8. SAM support

Please refer to [10] for more information on the SAM code.

8.1 Minor enhancements and bug fixes

Two minor items necessitated timely development to enable the continuation of NEAMS-funded SAM research:

- PR #23834 added support for boundary names that start with digits but later include non-numerical characters. Previously these names were being incorrectly parsed as numeric IDs. This fix enabled use of these names in inputs with `NodalScalarKernels`.

- PR #25245 responded to a SAM developer request to handle empty vector parameters differently. The default for vector parameters was to consider the lack of input as an input of an empty vector. This was modified to consider it as no input. This necessitated patches over much of the MOOSE-based application ecosystem.

9. Conclusion

This report highlighted various improvements made to the MOOSE framework in direct support of NEAMS applications. Some of the highlights include the addition of new Times and Positions systems, a redesign of mechanical contact constraints to enable the augmented Lagrange algorithm, an overhaul of the restart system, and an incorporation of p-refinement into MOOSE from libMesh. 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, 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.

[6] 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.

[7] C. Lee, Y. S. Jung, H. Park, E. R. Shemon, J. Ortensi, Y. Wang, V. M. Laboure, and Z. M. Prince, "Griffin software development plan," Tech. Rep. ANL/NSE-21/23, INL/EXT-21-63185, Idaho National Laboratory and Argonne National Laboratory, 2021.

[8] 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.

[9] 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.

[10] 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 Laboratory, Argonne, IL (United States), 2021.