INL/EXT-21-64493

NEAMS-Multiphysics Technical Assistance in FY-21

August 2021

Alexander Lindsay[1], Fande Kong[1], Guillaume Giudicelli[1], Robert Carlsen[1], Roy Stogner[1], and Andrew Slaughter[1]

[1] *Computational Frameworks, Idaho National Laboratory, Idaho Falls, Idaho 83415*

Idaho National Laboratory

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

INL/EXT-21-64493

NEAMS-Multiphysics Technical Assistance in FY-21

Alexander Lindsay[1], Fande Kong[1], Guillaume Giudicelli[1], Robert Carlsen[1], Roy Stogner[1], and Andrew Slaughter[1]

[1]Computational Frameworks, Idaho National Laboratory, Idaho Falls, Idaho 83415

August 2021

Idaho National Laboratory
Computational Frameworks
Idaho Falls, Idaho 83415

http://www.inl.gov

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

Page intentionally left blank

ACRONYMS

AD	automatic differentiation
BEA	Battelle Energy Alliance
DOE	U.S. Department of Energy
HFEM	hybrid finite element method
HLLC	Harten, Lax, Van Leer Contact
JFNK	Jacobian-Free Newton-Krylov
LM	Lagrange Multiplier
MOOSE	Multiphysics Object Oriented Simulation Environment
NCP	nonlinear complementarity problem
NEAMS	Nuclear Engineering Advanced Modeling and Simulation
PETSc	Portable Extensible Toolkit for Scientific Computation
PJFNK	Preconditioned Jacobian-Free Newton-Krylov
QUICK	Quadratic Upstream Interpolation for Convective Kinematics
SLEPc	Scalable Library for Eigenvalue Problem Computations
THM	Thermal Hydraulics Module

ABSTRACT

The Multiphysics Object Oriented Simulation Environment (MOOSE) [1] is a massively parallel finite-element/volume package for multiphysics simulation in science and engineering. The package focuses on providing rapid-development capabilities for engineering applications by leveraging well-built features from libMesh [2] and the Portable Extensible Toolkit for Scientific Computation (PETSc) [3]. Fiscal year 2021 (FY-21) was the first year with funding dedicated to supporting MOOSE-derived applications relevant to the Nuclear Engineering Advanced Modeling and Simulation (NEAMS) program. In this report we outline the work done to support NEAMS applications such as BISON, Griffin, Pronghorn, and System Analysis Module (SAM).

ACKNOWLEDGEMENTS

This work was funded by the U.S. Department of Energy (DOE) NEAMS program and made use of the resources of the High Performance Computing Center at Idaho National Laboratory. This manuscript was authored by Battelle Energy Alliance (BEA), LLC under Contract No. DE-AC07-05ID14517 with the U. S. DOE. The U. S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. 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 U. S. Government purposes.

Page intentionally left blank

CONTENTS

FIGURES

Page intentionally left blank

1. INTRODUCTION

2. BISON Support Tickets

2.1 Normal Field Consistency for Mortar Constraints

Mortar-segment meshes are constructed using projections based on interpolated nodal normals, where the nodal normals are computed on the secondary face. This nodal normal interpolation results in a continuous field. Prior to FY-21, however, the normal field used in mortar constraint objects was based only on the current mortar-segment element. This implementation allowed the possibility of discontinuous normals between mortar-segment elements. In general, more discontinuity produces a less robust nonlinear solve. To increase robustness and to create consistency with mortar mesh generation, in FY-21 we moved mortar constraint objects to use interpolated nodal normals instead of the traditional normal field generated by reinitializing the mortar-segment element. Users report this change leads to more efficient and robust solves on realistic contact problems.

2.2 Improved Mortar Thermal Contact Jacobian

We added computation of the gap dependence on displacement degrees of freedom in the mortar thermal contact object. The resulting Jacobian for temperature dependence on displacement degrees of freedom is perfect when tested against finite difference approximation of the Jacobian for a simple test case. Users report this change allows convergence of thermomechanical models that previously did not converge.

2.3 Variationally Consistent Mechanical Contact

Before FY-21, the zero-penetration mechanical contact constraint was enforced using a nonlinear complementarity problem (NCP) function that took node-wise Lagrange Multiplier (LM) and node-wise gap arguments even when using a mortar method. However, according to [4], using a node-wise LM and an **integrated** gap yields a variationally consistent method. Consequently, we implemented a two-pass constraint system in which a weighted/integrated gap was computed and stored in the system residual/Jacobian during the mortar-segment element mesh

loop and then queried in a node-wise constraint object where the NCP function was applied. This variationally consistent method appears very similar to the old node/point-wise method with the exception that the gap used as the argument to the NCP function contains much more information (due to the integration). The work conducted for this issue was later streamlined by the work outlined in Section 2.4.

2.4 Simplified Mortar Mechanical Contact Constraints

This work was a follow-on to that outlined in Section 2.3. In order to develop weighted frictional mechanical mortar contact constraints, we developed a `post` method for mortar constraints that executes after looping over the mortar-segment element mesh. For mortar mechanical contact constraints, the `post` method loops over degree-of-freedom objects (nodes when the Lagrange Multiplier is discretized with a dual basis) and applies NCP function(s). This implementation easily allows frictional constraints to query information from zero-penetration constraints, enabling the results shown in Figure 1 and Figure 2.

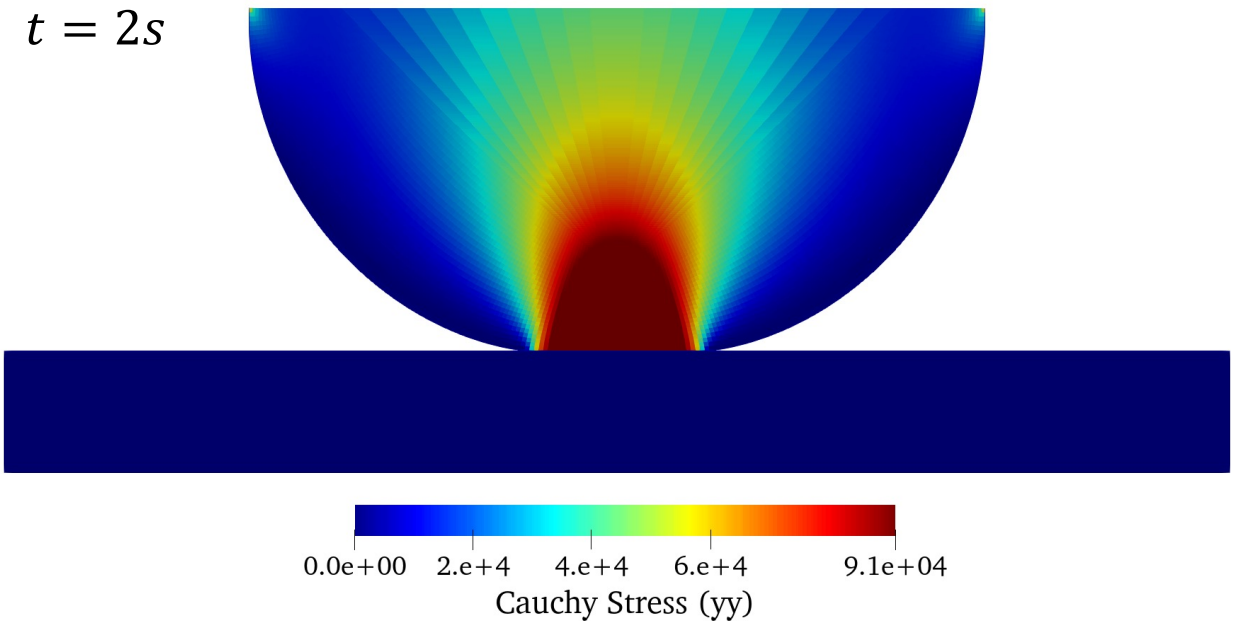
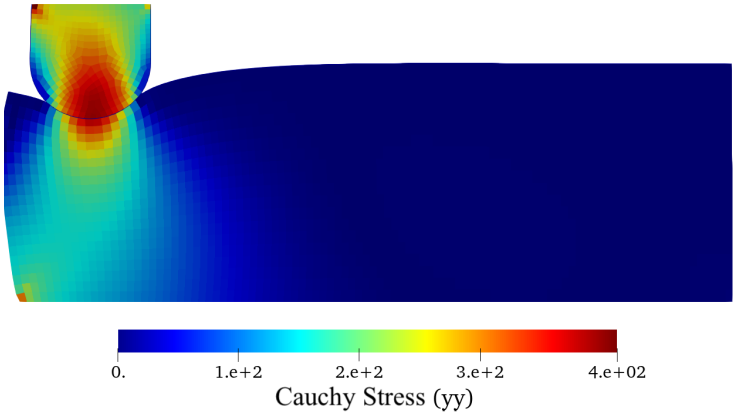
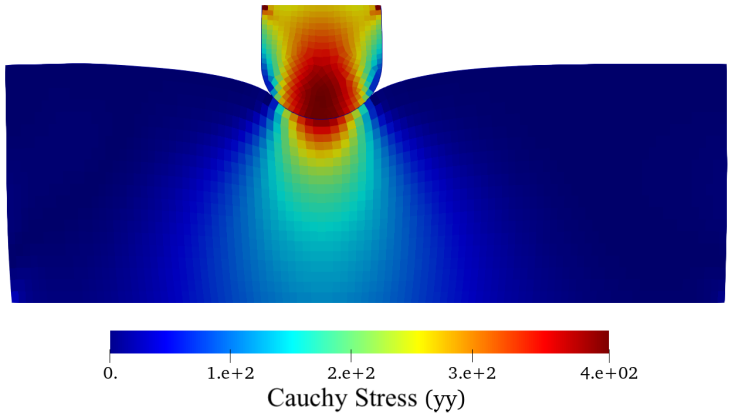


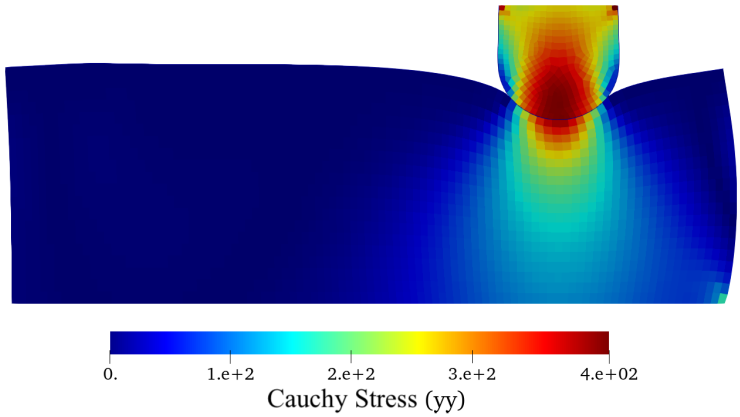
Figure 1: Hertzian contact of cylinder on a plane with the y directional Cauchy stress (σ_{yy}) shown.



(a) $t = 2$s



(b) $t = 5$s



(c) $t = 8$s

Figure 2: Stress fields at three different time steps for an iron drawn across a surface. The y directional Cauchy stress ($\boldsymbol{\sigma}_{yy}$) is shown in all cases.

2.5 Increase Robustness of ReinitElemPhys and ReinitNeighborPhys

Over FY-21, several users complained about failed point inversion warnings when running displaced mesh calculations. While the failed point inversions did not impact simulation result quality, the issue justifiably made users concerned about their problem setup and solution. We fixed this issue by no longer calling reference and displaced problem `reinit*Phys` routines together for the same provided physical points to invert. Instead, framework code that called `reinit*Phys` methods was modified to call only with the appropriate problem object, (e.g., `FEProblemBase::reinit*Phys` when a residual object is operating on the reference mesh and `DisplacedProblem::reinit*Phys` when the object is operating on the displaced mesh).

2.6 Eliminate Memory Requirements of AD Stateful Material Properties

With the help of a user-provided input, we discovered a bug in which automatic differentiation (AD) material properties that had requests for old or older property states were triggering tremendous memory use. This was because we were doing storage of current property values as AD types for all the elements of the mesh. To address this issue, when we swapped current evaluations of properties into storage, we immediately converted the property evaluation from AD to its non-AD equivalent, dropping the associated derivative array. This reduced associated memory use by a factor of the array size, which for most MOOSE simulations is 50.

3. Griffin Support Tickets

3.1 Correct Lower Dimensional Element Areas

We recently added support for the hybrid finite element method (HFEM) which is useful for nodal diffusion neutronics calculations. HFEM relies on flux computations on element faces and requires accurate computation of element face/lower dimensional element areas. When adding support for HFEM, we discovered that area computations were only accurate for cartesian coordinate systems. We have since updated the lower dimensional element area computation code to support cartesian, 2D-axisymmetric, and 1D R-spherical coordinate systems.

3.2 Force the Execution of UserObjects after the Execution of AuxiliaryKernels

In order to allow a different ordering of user objects in Griffin, the Griffin team made a pull request to MOOSE to make the dependency resolution between user objects and auxiliary kernels cognizant of the time of execution, such as at the beginning of every time step or only at the end of the simulation. Unfortunately this pull request to MOOSE led to a test failure in BISON, for the computation of UO_2 eigenstrains. Because the Griffin team member in charge did not have access to BISON, we resolved the issue for them.

The BISON test output was dependent on the execution of a user object before the auxiliary kernel that used its output. With the pull request to MOOSE, the dependency resolution was correctly executing the user object after the auxiliary kernel, as their execution flags were different. In order to fix the test without breaking the continuous integration (e.g., having BISON or MOOSE failing tests at any point in time), we added the functionality in MOOSE to force the execution of user objects after the auxiliary kernels. This allowed us to fix the test in BISON with the updated version of MOOSE. Then the pull request made by the Griffin team member to MOOSE passed the BISON tests and was subsequently merged.

3.3 Mesh Sub-generators to Enable Complex Mesh Generation

The internal MOOSE mesh generation capabilities enable users to create meshes for any of a number of basic domain shapes, controlling domain and mesh parameters via input file. Next, these meshes can be modified and combined together via objects specified in other input file subsections. This allows users to create arbitrarily large trees of MOOSE mesh generator dependencies at runtime so they can construct meshes for complex simulation domains.

The flexibility of this interface, however, is also a source of difficulty: to construct a mesh in this fashion, parameters for every sub-mesh must be specified independently, redundantly, and correctly. A support request was made for MOOSE to add the ability to allow a custom mesh generator to create its own "sub-generator" objects in C++ code, specifying sub-generator parameters based on higher level generator parameters without the need for user input. Application-specific `MeshGenerator` subclasses could then take the micromanagement of mesh subcomponent con-

struction out of user hands, and construct detailed meshes of known or parametrically defined domains based on only a simplified set of input parameters.

Although this sort of programmatic sub-generator instantiation was already technically possible within MOOSE, the mechanism for doing so was not obvious, not documented, not tested or regression-tested, and not necessarily supported. We encapsulated this capability into two new `addMeshSubgenerator` APIs in the `MeshGenerator` class, one of which specifies subgenerator parameters via a standard MOOSE `InputParameters` object, and the other of which builds input parameters on the fly from a variadic argument list (for easy "one-liner" submesh creation). New tests (including one with a new `MeshGenerator` subclass test object) have been added to the MOOSE CI suite to verify both versions of the capability, and description of the subgenerator capability has been added to the documentation describing how to create custom mesh generator subclasses.

3.4 Avoid Creating Unnecessary Vectors and Matrices

Previously, the system matrix was allocated regardless of whether the solve was matrix-free or not (for example, a Jacobian-Free Newton-Krylov (JFNK) solve does not require a system matrix). In addition, solution vectors for the old and older states of the solution were allocated regardless of whether the solve was a transient solve or not (for example, when the solve is steady state). For applications that often require a significant number of degrees of freedom (e.g., Griffin), these extraneous allocations can be quite costly. Now, the system matrix is only allocated when it is required and the old and older solutions of both the nonlinear system and the auxiliary system are allocated only on request.

3.5 Support Maps in Input Parameters

Griffin used to have many input parameters that looked like pairs of vectors, such as `initial_isotopes` and `initial_densities`. We added support for input parameter maps such that the pair of vectors can be be converted to a map. For instance, after addition of this support, the isotope input file syntax now appears as `initial_isotopes = 'H1 0.1 O16 0.1 U235 0.1'`.

3.6 Custom Face Quadrature Rules for Different Subdomains

Previously, in Pull Request #15470, the ability was added for users to specify block-specific custom quadrature orders. This functionality, however, was limited to volume/elemental quadrature rules only. The face quadrature rules were still global and not allowed to be block-specific. This was Issue #15472. While ambiguity along the inter-block boundaries has historically been resolved by selecting based on the lower element ID, element numbering does not guarantee lower element IDs along a block interface are all inside one of the two blocks. This problem is exacerbated by mesh partitioning, which can even cause this numbering to be different from run to run for a single, static simulation input.

In order to implement custom face quadrature rules per block, an implementation was created that determines and uses the highest order quadrature specified for all blocks a face is touching (see Pull Request #15790). This allows users higher order, block-specific custom quadrature rules with face integration along inter-block boundaries without element iteration or partitioning-based inconsistencies.

3.7 Eigenvalue Executioner Refactors and Improvement

Eigenvalue solvers are fundamentally crucial in multiphysics simulation. For example, the criticality calculations of neutron-transport equations using an eigenvalue solver provide insight into the sustainability of nuclear reactors, which, in turn, helps engineers operate control rods. Many modern eigenvalue-solving techniques exist, but it would be costly to implement them from scratch. Instead, we leveraged these modern techniques' implementations from Scalable Library for Eigenvalue Problem Computations (SLEPc) by developing a pluggable eigenvalue system in the MOOSE framework.

The pluggable eigenvalue system was developed in previous years. In FY-21, we refactored the core part of the system, the eigenvalue executioner, to introduce new features and fix critical issues. The development activities are described as follows:

1. Support Picard iteration in the eigenvalue solvers between multiple applications for multiphysics simulation. For such a goal, the eigenvalue executioner was refactored to separate the nonlinear power iteration from the Newton method. At the beginning of the simulation,

the Newton method will take an initial vector from the nonlinear power iteration. During Picard iteration, the Newton method takes the solution of the previous Picard iteration as an initial guess. Otherwise, if the Newton method always uses a nonlinear power iteration as its initial guess, then Picard iteration can not converge.

2. Enhance user interfaces to solve users' particular applications. We improved the system to display user-informed progress messages during code execution. The support of colorful linear and nonlinear residual monitors was also added for users' convenience.

3. Clean up input-file executioner block by removing unnecessary parameters.

4. Preconditioned Jacobian-Free Newton-Krylov (PJFNK) was chosen as the default eigenvalue solver option because of its good performance for different problems. There was an issue when using PJFNK without a power iteration because SLEPc was not set up correctly. We fixed that by setting the necessary SLEPc options to trigger a correct solver setup.

5. In a parallel simulation, a distributed vector is passed from petsc, and in addition to the local values, we need to get extra ghosting values for residual and Jacobian calculations. We used an API, `VectorLocalize`, from libmesh to gather all values to every processor. That worked well for small problems using a small number of processors but could not scale to large problems. We fixed the issue using the libmesh system update capability, where only local and ghost values are gathered, and the algorithm is scalable.

3.8 HMG Preconditioner Option Auto-selection

It is challenging to solve a large nonlinear system of algebraic equations arising from the discretization of multiphysics problems. Linear solvers often dominate the simulation time. A scalable, efficient preconditioner with optimal options is required to speed up the linear solution. The development of an efficient preconditioner can be motivated by the structure of an underneath application. For example, for neutron transport calculations, many unknowns are associating with each mesh node. An efficient preconditioner can be designed by considering that structure. Previously, we developed a preconditioner, PCHMG, in PETSc precisely for this purpose. HMG works well only when suitable petsc options are chosen. For neutron simulations, memory usage is the

biggest concern, and in FY-21, we automatically chose memory-efficient options, for example, for interpolation and restriction operators when HMG is employed. That enables memory-efficient neutron transport simulations.

3.9 Programmatic Heap Profiling

Memory usage is an essential aspect of multiphysics simulation. It is not easy to manually track down poor memory management in the code. Best practice is to use a profiling tool to track all memory usage and locations of allocations when the simulation is carried out. We might improve the code with the profiling results by either introducing a new algorithm or improving the existing algorithm. We opted to use a third-party tool, gperftools, to assist us with the task. The CPU profiling capability of gperftools had been previously integrated into MOOSE, and in FY-21, we added a programmatic heap profiling capability by adding environment variables such as `MOOSE_HEAP_BASE` and `HEAP_PROFILE_INUSE_INTERVAL`. `MOOSE_HEAP_BASE` is utilized to build file names of profiling results, and `HEAP_PROFILE_INUSE_INTERVAL` specifies how much increase in memory is allowed before outputting profiling results.

3.10 Memory Improvement in Matrix Assembly Class

A vector or matrix must be assembled during residual or Jacobian calculations before solving the corresponding algebraic system. In MOOSE, that is done in the `Assembly` class. Several flag vectors are used to indicate whether or not we need to assemble values for specific coupling entries. By default, the flag vectors consider full coupling, leading to memory spikes for applications with many nonlinear variables on each mesh element. We eased this issue by refactoring the API to make diagonal couplings as the default setting. For a case we checked, the memory usage in this particular code spot was reduced from 151 MB to 16.5 kB.

4. Pronghorn Support Tickets

4.1 Global AD Indexing

Finite volume methods can sometimes require large stencils of information (e.g., neighbors of neighbors information). When dealing with such large stencils it can be difficult to develop a local

numbering scheme or pre-initialize enough data to perform the required computation. In order to accommodate these large stencils, we developed a new method for indexing dual numbers for computing Jacobians. Instead of using an abitrary local numbering scheme, we index according to the global degree of freedom. This eliminates any possibility of ambiguity about what degrees of freedom a given `ADReal residual` depends on and makes assembly of the system Jacobian straightforward and easy to maintain.

Global AD indexing was initially developed to support Pronghorn finite volume methods, but it has since been applied in the Thermal Hydraulics Module (THM) because of the ease in which it allows coupling between different system components. The capability is also being leveraged in current development of an evaluation system that largely erases the need for dependency tracking.

4.2 Slope Limiters for Finite Volume

To implement advection schemes for the fully compressible Euler or Navier-Stokes equations, we developed a `Limiter` system in MOOSE which is used to limit interpolations of cell center quantities to faces. Some of the limiters that have been implemented to-date include first-order upwind, central differencing, Quadratic Upstream Interpolation for Convective Kinematics (QUICK), second-order upwind, Van Leer, and minmod. These limiters have been incorporated into Kurganov-Tadmor computation of advective fluxes and will likely be incorporated into Harten, Lax, Van Leer Contact (HLLC).

4.3 Restarting Scalar Variables

Closed system incompressible flows require some specification of pressure to make the system nonsingular. One way to do this pressure specification is to use an LM to set the system mean pressure. This method is the most commonly employed method for MOOSE's incompressible finite volume implementation. When a user attempted to use the LM method, they discovered that a simulation restarted from a previously converged solve stored in Exodus did not register as converged when it should have. This defective behavior was due to incorrect initialization of the LM variable. The issue was remedied by using built-in libMesh capability for reading scalar variable results from Exodus and deleting the old defective MOOSE code for scalar initialization.

4.4 Ensure Correct Jacobians for Finite Volume with Auxiliary Variables

It was possible to incorrectly seed dual numbers in finite volume calculations if auxiliary variables were present. To counter-act this, we have added checking of the libMesh system number in order to ensure that we are only seeding dual numbers for nonlinear variables, ensuring accuracy of the system Jacobian.

4.5 Application Level Option for Controlling New Nonzero Behavior

We added an application-level option for controlling behavior when PETSc encounters new nonzero allocations. The default in MOOSE is to not emit an error when PETSc is forced to allocate new memory when building the system matrix. This default enables computations on displaced meshes because it is extremely difficult to accurately compute sparsity patterns for many or even a single timestep since element-to-element coupling throughout a nonlinear solve may change even from nonlinear iteration to iteration. However, for applications that do not often use displaced meshes, such as Navier-Stokes/Pronghorn, it is valuable to change the default behavior to error on new nonzero allocations as they indicate a fundamental problem in the application or framework code. This allows developers to track down possible ghosting errors and to eliminate the very high computational expense of these new nonzeroes.

5. System Analysis Module (SAM) Support Tickets

5.1 Automatic Scaling, Multiapps, and Restart

A user reported they could not do automatic scaling in sub-applications in conjunction with restart. To ameliorate this bug we added the capability in libMesh to write out vector parallel typing (e.g., `PARALLEL`, `GHOSTED`, or `SERIAL`) to checkpoint files which enabled writing and reading (upon restart) of automatic scaling factor vectors.

6. Workbench Support Tickets

6.1 Definition Formatter Enhancements

We removed declarator requirements, made occurrences unlimited, and pointed `ChildAtLeastOne` rule paths to parent rather than value contexts. These changes made NEAMS workbench input validation and auto-completion assistance of MOOSE applications more reliable.

6.2 Add Custom FunctionExpression Type to Some Contexts

We have added `FunctionExpression` designations to parsed objects, allowing NEAMS Workbench input validation for all parameters.

7. Miscellaneous Support Tickets

7.1 Make Cohesive Zone Models More Robust

We added a new data member, `_reinit_displaced_neighbor`, to distinguish between cases where element faces have to be reinitialized on displaced meshes for objects like integrated boundary conditions, and where element **neighbor** faces have to be reinitialized on displaced meshes for objects like discontinuous Galerkin or interface kernels. Previously, displaced integrated boundary conditions were triggering reinitialization of neighbor faces on displaced meshes in cohesive zone contexts, leading to failed point inversions and a lack of user confidence.

7.2 Consider Off-diagonals for Automatic Scaling

Automatic scaling can bring the Jacobian scale or residual entries of individual variables to similar levels. It can be viewed as a kind of nonlinear preconditioning. Until FY-21, automatic scaling was considered only with on-diagonal entries of the Jacobian. When the on-diagonal is zero, as in saddle-point problems like incompressible Navier-Stokes, an on-diagonal-based scaling strategy cannot work. In that vein, we added the capability to consider off-diagonal Jacobian entries in the automatic scaling computation. This capability can be triggered by setting `off_diagonals_in_auto_scaling = true` in the `Executioner` input file block. We tested this im-

plementation on a lid-driven incompressible cavity problem and saw a conditioning number improvement of 17 orders of magnitude compared to the on-diagonal based strategy.

7.3 Protect Against Zero-Size Column Indices when Doing AD

A tensor mechanics user reported errors out of BLAS when performing an AD computation with periodic boundary conditions. The error was caused by using an AD object that had no dependence on the nonlinear degrees of freedom in conjunction with periodic boundary conditions, which require constraint enforcement in libMesh. BLAS errors are produced if libMesh constraint enforcement is called with a local Jacobian that has zero columns. We removed this error by simply returning out of AD derivative processing routines when there is no nonlinear degree of freedom dependence of the passed-in residual(s).

7.4 Resolve Interface Material Dependencies

We added sorting of interface materials based on requests with `getMaterialProperty` and supply with `declareMaterialProperty`. This resolved a bug reported by one of our NEAMS application developers, where a material property depending on another was seeing wrong values for the dependency property.

7.5 Other Miscellaneous Tickets

- Pull Request #16619: Programmatically set the `DofMap` for `RelationshipManagers` so that systems beyond the typical nonlinear and auxiliary systems can have algebraic ghosting available for their computations.

- Issue #16395: Automatically force a full Jacobian when using AD with global indexing. This is because global AD indexing (introduced in Section 4.1) fills in the Jacobian matrix irrespective of the user-specified coupling matrix. Consequently, unless we allocate a full a Jacobian, users may experience new memory allocation errors out of PETSc.

- Issue #16360: Avoid creating vector copies in `ArrayKernels`. This change has sped up `ArrayKernel` based computations in Griffin by 20%.

- Pull Request #15338: We put a lot of effort into increasing the robustness of geometric and algebraic ghosting on displaced and distributed meshes, including other advanced features such as periodic boundary conditions. This has greatly reduced the number of user error reports related to ghosting.
- Issue #13609: We created a `ResidualObject` class that allows significant reduction in code duplication between residual computing classes such as `Kernel`, `BoundaryCondition`, `DGKernel`, `InterfaceKernel`, etc.
- Issue #16776: We now support the addition of user-requested partitioners to mesh generators. Previously, if a user requested a specific partitioner but constructed their mesh using mesh generators, then their partitioner request would be totally disregarded.

REFERENCES

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

[2] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, "libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations," *Engineering with Computers*, vol. 22, no. 3-4, pp. 237–254, 2006.

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

[4] B. Wohlmuth, "Variationally consistent discretization schemes and numerical algorithms for contact problems," *Acta Numerica*, vol. 20, pp. 569–734, 2011.