INL/EXT-20-59837-Revision-0

NEAMS-Multiphysics MOOSE End Year Framework Activities FY20

February 2024

Changing the World's Energy Future

Alexander D Lindsay, Robert W Carlsen, Derek R Gaston, Andrew E Slaughter, Fande Kong, Jason Mathew Miller, Roy Stogner, Cody J Permann

Idaho National Laboratory

INL/EXT-20-59837-Revision-0

NEAMS-Multiphysics MOOSE End Year Framework Activities FY20

Alexander D Lindsay, Robert W Carlsen, Derek R Gaston, Andrew E Slaughter, Fande Kong, Jason Mathew Miller, Roy Stogner, Cody J Permann

February 2024

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

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

INL/EXT-20-59837

NEAMS-Multiphysics MOOSE End Year Framework Activities FY20

Alexander Lindsay
Fande Kong
Robert Carlsen
Jason Miller
Andrew Slaughter
Roy Stogner
Cody J. Permann
Derek R. Gaston

September 2020

INL/EXT-20-59837

NEAMS-Multiphysics MOOSE End Year Framework Activities FY20

Alexander Lindsay
Fande Kong
Robert Carlsen
Jason Miller
Andrew Slaughter
Roy Stogner
Cody J. Permann
Derek R. Gaston

September 2020

Idaho National Laboratory
Modeling and Simulation Department
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

ABSTRACT

The Multiphysics Object Oriented Simulation Environement (MOOSE) is a general finite element package meant for high performance solution of multiphysics problems in science and engineering. In this report, we present additions and enhancements to MOOSE funded by the Nuclear Engineering Advanced Modeling and Simulation (NEAMS) program. NEAMS-funded MOOSE framework library improvements include: expansion of support for multi-level multi-application restart; enhancement of coupling between native MOOSE applications and external libraries; addition of a sparse Automatic Differentiation (AD) container enabling non-local degree of freedom coupling; block-specific quadrature rules; further development of an eigenvalue executioner; faster setup of periodic boundary conditions; and creation of a mesh meta-data system streamlining simulation startup. Besides developing the framework library, NEAMS funds were used to overhaul a swath of important MOOSE infrastructure in order to substantially improve user experience. These critical infrastructure changes included adapting MOOSE to python 3, improving the test harness to catch race conditions, and most importantly transitioning MOOSE to use Conda, a globally known packaging system, that greatly eases software adoption.

ACRONYMS

AD	Automatic Differentiation
API	Application Programmer Interface
BEA	Battelle Energy Alliance
DOE	U.S. Department of Energy
MOOSE	Multiphysics Object Oriented Simulation Environement
NEAMS	Nuclear Engineering Advanced Modeling and Simulation
ORNL	Oak Ridge National Laboratory
PETSc	Portable Extensible Toolkit for Scientific Computation
SLEPc	Scalable Library for Eigenvalue Problem Computations

The Multiphysics Object Oriented Simulation Environement (MOOSE) is a general finite element package meant for high performance solution of multiphysics problems in science and engineering. In this report, we present additions and enhancements to MOOSE funded by the Nuclear Engineering Advanced Modeling and Simulation (NEAMS) program. NEAMS-funded MOOSE framework library improvements include: expansion of support for multi-level multi-application restart; enhancement of coupling between native MOOSE applications and external libraries; addition of a sparse Automatic Differentiation (AD) container enabling non-local degree of freedom coupling; block-specific quadrature rules; further development of an eigenvalue executioner; faster setup of periodic boundary conditions; and creation of a mesh meta-data system streamlining simulation startup. Besides developing the framework library, NEAMS funds were used to overhaul a swath of important MOOSE infrastructure in order to substantially improve user experience. These critical infrastructure changes included adapting MOOSE to python 3, improving the test harness to catch race conditions, and most importantly transitioning MOOSE to use Conda, a globally known packaging system, that greatly eases software adoption.

1 Introduction

NEAMS support for MOOSE covered a breadth of tasks in FY20. Additions and enhancements for capability and performance of MOOSE-based applications are outlined in 2. Changes to MOOSE infrastructure that improve testing and uptake of the MOOSE ecosystem are outlined in 3.

2 Framework Library Improvements

2.1 Multilevel Multiapp Restart Support

During the Picard iteration of the multilevel `MultiApp` system, the initial state of sub-applications is stored. Historically, the initial state is restored when the sub-applications are called again in the next Picard iteration. In general, this strategy works pretty well; however, sometimes it may be beneficial to use the previous simulation's solution as the initial guess for the current simulation in the sub-applications. As such, we added an input file option `keep_solution_during_restore` to allow the previous simulation solution to be reused as the initial guess of the current simulation in the multilevel multi-app restart process. The parameter is required only in the master application input file. During the restoration process, the solutions are stored recursively in the multilevel `MultiApp` system. After the restoration, the stored solutions are copied as initial guesses to the sub-application simulations. The initial guesses are also synced in parallel so that the ghosted values are up-to-date.

2.2 External Application Coupling Demo

The MOOSE framework can couple not only MOOSE native applications but also external applications. For connecting external applications, an application wrapper technique is used, where the wrapper behaves as a proxy for the external application. During FY20, we improved an example, demonstrating the use of the application wrapper to couple an external simulation. The example couples an external Portable Extensible Toolkit for Scientific Computation (PETSc) [1] simulation with a native MOOSE application. In the example, the external application can work as either a master application or a sub-application. Data store and restore capability was implemented to enable Picard iteration between the external application and the native MOOSE application. A customized time stepper was added to query the time step size from the external application. In this way, the external application can drive the selection of time step size. Regression tests were added to ensure the code continues operating smoothly through future MOOSE development. Following creation of this example, a team at Oak Ridge National Laboratory (ORNL) successfully coupled a neutron calculation code, MPACT, with MOOSE animals such as BISON.

2.3 Automatic Differentiation for Scalar Kernels

A significant refactoring of MOOSE's AD system was completed in March 2020. Prior to the refactor, MOOSE's AD system required users to work with C++ templates. While use of templates maximized the speed of AD code, it presented a significant barrier to entry for new users with limited C++ background. Additionally, it put an onerous burden on MOOSE framework developers who had to ensure that data was consistently synced between different template instantiating for residual and Jacobian computation. However, addition of a static Boolean `do_derivatives` member to the `Dual Number` class at the center of MOOSE's AD system enabled removal of templates. The `do_derivatives` member is toggled to `false` at the beginning of residual computations, and toggled to `true` when performing Jacobian computations. In this way single AD kernel objects can be used for computation of both residuals and Jacobian; in the former case, the `false` value for the `do_derivatives` member prevents the AD system from performing expensive and unnecessary chain rule derivative operations. In terms of performance, for the worst case in which a user is using a matrix-free approximation of the Jacobian, a slow-down of 1.5-2x is observed after the AD refactoring since there is non-zero overhead when checking the `do_derivatives` member in every `Dual Number` operation. However, for the solve type with which AD is intended to be used, Newton, there is no observable difference in run-time between the original AD implementation and the refactored implementation. Considering the negligible performance effects, the improved user experience with AD, and the vast reduction in maintenance cost, the refactor was considered well worth the effort. During the last MOOSE training given in June 2020, users were able to pick-up the AD system much more readily than in previous training's. Moreover, as a measure of AD maintenance cost, the refactor reduced the number of lines of AD code by 4,000, a significant fraction of the total AD code size.

In addition to improving the user experience, significant new sparse AD capability has been added to MOOSE. MOOSE sparse AD relies on a new derivative container class called `SemiDynamicSparseNumberArray`, which was introduced in a dependency library, MetaPhyiscL. The `SemiDynamicSparseNumberArray` class contains an array data member for tracking the degree of freedom index associated with each partial derivative of a `DualNumber's` function value. This index array opens the door to much greater flexibility in the AD system. For instance, MOOSE previously relied on an arbitrary local indexing scheme in order to track which value in a `DualNumber's` partial derivative vector corresponded to which global degree of freedom index; this knowledge is necessary for correct assembly of the global Jacobian matrix. However, with the ability to hold indices in memory in `SemiDynamicSparseNumberArray`, a local indexing scheme is no longer necessary. Instead each partial derivative is associated with the global degree of freedom index directly. Consequently, partial derivatives of functions with non-local dependence, e.g. functions that depend on information beyond the current finite element or volume, can be easily computed and tracked. This is invaluable for things like reconstructing gradients in MOOSE's new finite volume formulation; these gradients require at a minimum two layers of geometric information beyond the current finite element or volume. Using a local indexing scheme for such a large stencil would be intractable; however, global indexing makes Jacobian computation for the reconstructed gradient simple and straightforward. Finally, this global indexing capability is invaluable for computing Jacobians for `ScalarKernels`, where the `ScalarKernel` function may depend on an arbitrary number of the simulation's degrees of freedom.

2.4 Block Specific Quadrature

Support was added enabling users to specify different quadrature (integration) orders for individual mesh subdomains. Performing simulations with higher order quadrature rules requires significantly longer run-time – increasing proportionally with the number of quadrature points in the worst cases. Computationally expensive higher order quadrature rules can now be isolated to run only in portions of the mesh where the physics and simulation details require it. In order to take advantage of this functionality users simply list the desired quadrature orders for each mesh subdomain in their input files.

2.5 Eigenvalue Executioner Enhancements

The Scalable Library for Eigenvalue Problem Computations (SLEPc) [2] based eigenvalue executioner in MOOSE was designed to solve linear and nonlinear eigenvalue problems. To handle recent challenges, especially in neutron transport calculations, we furthered the eigenvalue executioner capability. Firstly, we added a PETSc-option databases in MOOSE for enabling Picard iteration, which is essential for coupling neutronics with other physics such as thermodynamics. The PETSc options used to setup eigenvalue solvers or nonlinear solvers are stored in this database when stepping into a sub-application. These options are restored correctly when the sub-application simulation is completed, and when the master-application needs to execute its solver. Compared to a conventional method, the database approach streamlines processing of solver options. Picard-iteration regression tests were added to MOOSE, demonstrating use of the eigenvalue executioner in either sub-application or master application.

Secondly, shell-matrix support was implemented. The SLEPc linear eigensolver Application Programmer Interface (API) requires some matrices for proper configuration. However, it can be difficult for many MOOSE applications to explicitly form these required matrices due to the complexity of the multiphysics residual functions or because of the high memory demand. To overcome this challenge, we introduced shell matrices in MOOSE, which compute matrix-vector operations solely using residual evaluations. This enhancement is critical when the problem is too complex to compute derivatives or too large to store all matrix entries. While the full linear operator is implemented using a shell matrix, the preconditioner is constructed using an explicitly formed matrix approximating the full operator.

Finally, we made the eigenvalue executioner and solver as consistent with MOOSE's traditional nonlinear solver as possible. The SLEPc package is treated as a dependent component of PETSc and managed via the command-line option `--download-slepc`. We enhanced the libmesh [3] `configure` script to autodetect SLEPc from the PETSc environment. We now automatically assign a negative sign to `EigenKernels` as if they were moved to the left-hand side of their equation description, which is consistent MOOSE's traditional nonlinear solve. Integrated boundary conditions are also now supported for eigenvalue simulations, increasing its viability for complex neutron transport simulations.

2.6 Mesh MetaData Store

A new system was added to MOOSE to better manage mesh related attributes during simulation setup. This enables easy access to "metadata" about the mesh or domain regardless of the conditions under which the simulation was started. BISON was the primary motivator of this new system, due to its desire to obtain fuel performance geometric information–such as pellet stack heights, cladding, and gap thicknesses–without having to search and interrogate the entire mesh. While this information is readily available when using MOOSE's on-the-fly `MeshGenerator` system, that system is not always used in cases such as "restart" or "recovery" modes, meaning that special case logic was being written in completely unrelated objects to accommodate the possibility of data coming from different locations under different start scenarios. The framework needed a consistent way to provide mesh metadata attributes to any object that may need them, including both `MeshGenerator` and `Action` systems.

This new `MetaData` system leverages the existing MOOSE backup/restore capability to store arbitrarily-typed attributes to a stateful data-store during startup. A separate checkpoint file is written/read, which contains any mesh related attribute that may be required during the startup phase of the simulation. The `MetaData` enhancement provides a first-class capability to all objects in the simulation, removing the need to code dynamic casts or special case logic to find mesh related parameters. This simplifies logic in BISON, Griffin, and other applications which make heavy use of the `MeshGenerator` or restart systems.

2.7 Faster Periodic Boundaries

Periodic boundary conditions are widely used in many applications, such as those using the phase-field module. The periodic boundary conditions setup can be costly since each boundary vertex needs to be paired with a boundary vertex on the other side of the domain via a geometric search. This geometric search used to be implemented using an $O(N^2)$ algorithm. A new $O(NlogN)$ tree-based algorithm was implemented during FY20. On a small test problem, this algorithm change sped up the simulation time by 2x. For larger, more realistic simulations, the new algorithm has sped-up periodic boundary condition setup by 10x-100x.

3 Infrastructure Improvements

3.1 Python 3 Upgrade

Language support for python 2 ceased on January 1, 2020. As such, in late 2019 all python tools within MOOSE were updated to python 3. This includes the testing system, the peacock graphical interface, the visualization tool chigger, the MooseDocs documentation system, as well as many other utilities. The transition required maintaining python 2 support for the critical systems (testing and documentation) until the transition was complete. As of early 2020, all tools have been upgraded and the use of python 2 is no longer supported.

3.2 MOOSE Configure System

A configure system based on GNU autotools was added, initially to support configuring MOOSE with different automatic differentiation containers. With the existence of MOOSE configure, users may now choose whether to use a non-sparse container, which may be optimal when running similar simulations and variable sets repeatedly, or a sparse container, which is optimal when the user wants to run a variety of types of problems in different dimensions and/or when coupling between variables is sparse. The latter container encouraged addition of another configure option to MOOSE: global vs. local indexing of derivatives. The technology behind the new global indexing capability is described in 2.3. Although MOOSE's configure system was initially created for controlling automatic differentiation settings, it can be used to control many other compile-time settings. For instance, MOOSE's configure system automatically detects whether a user's machine has the `libpng` library, and if it does, PNG output capabilities are enabled.

3.3 Conda and Build Infrastructure

We transitioned from our legacy MOOSE-environment compiler stack to a Conda based one, using the Civet continuous integration system. Creation of the Conda stack is triggered automatically by commits into the MOOSE repository. The Conda stack allows for MOOSE development within a known working environment. More critically, it dramatically increases the ease at which MOOSE can be picked-up by a new user since they are no longer required to build any of MOOSE's library dependencies. In addition to Conda, work continues to make Spack our compiler stack replacement in the build farm, where Conda cannot function properly due to platform-agnostic configuration.

3.4 TestHarness Scheduling

A known issue with the `TestHarness` has been that two tests can concurrently attempt to write the same output file. This was something that historically the `TestHarness` was not able to detect since it requires parsing and understanding MOOSE input files. To that end, a feature was added called `--pedantic-checks`. This feature forces tests in an individual test group to run serially, and then takes snapshots of the file system after each individual test is run; if a later test attempts to write to a file saved in the snapshot, the `TestHarness` will emit an error indicating a caught race condition, unless the user has specified proper prerequisites. Because file snapshotting is slow, it is only run on MOOSE's continuous integration system, Civet, and not on users' local machines. However, serial test running within a test group **is** enabled by default on all machines unless disabled through the `tests` spec option `parallel_scheduling = true`. This is useful to increase overall testing speed when those test spec files which have been thoroughly scrutinized for race conditions.

Acknowledgments

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

References

[1] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.11, Argonne National Laboratory, 2019.

[2] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[3] Benjamin S Kirk, John W Peterson, Roy H Stogner, and Graham F Carey. libmesh: a c++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers*, 22(3-4):237–254, 2006.