SoftwareX 11 (2020) 100430

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

MOOSE: Enabling massively parallel multiphysics simulation

Cody J. Permann [a,*], Derek R. Gaston [a,*], David Andrš [a], Robert W. Carlsen [a], Fande Kong [a], Alexander D. Lindsay [a], Jason M. Miller [a], John W. Peterson [c], Andrew E. Slaughter [a], Roy H. Stogner [b], Richard C. Martineau [a]

[a] Computational Frameworks, Idaho National Laboratory, Idaho Falls, ID, 83415, United States of America
[b] Institute for Computational and Engineering Sciences, The University of Texas at Austin, Austin, TX, 78712, United States of America
[c] Akselos Inc., 2101 West Blvd., Houston, TX, 77042, United States of America

ARTICLE INFO

Article history:
Received 20 September 2019
Received in revised form 17 February 2020
Accepted 18 February 2020

Keywords:
Framework
Finite-element
Parallel
Multiphysics
Multiscale

ABSTRACT

Harnessing modern parallel computing resources to achieve complex multiphysics simulations is a daunting task. The Multiphysics Object Oriented Simulation Environment (MOOSE) aims to enable such development by providing simplified interfaces for specification of partial differential equations, boundary conditions, material properties, and all aspects of a simulation without the need to consider the parallel, adaptive, nonlinear, finite element solve that is handled internally. Through the use of interfaces and inheritance, each portion of a simulation becomes reusable and composable in a manner that allows disparate research groups to share code and create an ecosystem of growing capability that lowers the barrier for the creation of multiphysics simulation codes. Included within the framework is a unique capability for building multiscale, multiphysics simulations through simultaneous execution of multiple sub-applications with data transfers between the scales. Other capabilities include automatic differentiation, scaling to a large number of processors, hybrid parallelism, and mesh adaptivity. To date, MOOSE-based applications have been created in areas of science and engineering such as nuclear physics, geothermal science, magneto-hydrodynamics, seismic events, compressible and incompressible fluid flow, microstructure evolution, and advanced manufacturing processes.

Code metadata

Current code version	0.9.0-pre
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_293
Legal Code License	GNU LGPL 2.1
Code versioning system used	git
Software code languages, tools, and services used	C++, Python
Compilation requirements, operating environments & dependencies	GCC 4.8.4+ Clang 3.5.1+ Memory: 16GB+ Disk: 30GB+ OS: Mac OS 10.13+, Linux (POSIX) Deps: MPI, PETSc, Hypre, libMesh
If available Link to developer documentation/manual	https://mooseframework.org
Support email for questions	moose-users@googlegroups.com

1. Motivation and significance

As computing capability increases, scientists are exploring computational solutions to complex problems involving multiple scientific domains. These multiphysics problems tie together

* Corresponding authors.
E-mail addresses: cody.permann@inl.gov (C.J. Permann), derek.gaston@inl.gov (D.R. Gaston).

https://doi.org/10.1016/j.softx.2020.100430

many disciplines and involve collaborators with specific knowledge. Traditionally, multiphysics simulation was achieved by connecting individual simulation tools built by various researchers, which is time-consuming, error-prone, and problematic for taking advantage of modern parallel computing. The Multiphysics Object Oriented Simulation Environment (MOOSE) [1] focuses on a rich and flexible open-source environment for the development of such software. The framework provides a plug-in infrastructure that simplifies definitions of physics, material properties, multiphysics coupling, and postprocessing. The design allows developers to focus on their scientific endeavor without needing to understand the intricacies of modern parallel computing. In addition, uniformity in problem specification leads to maximizing reuse. MOOSE-based projects can share capabilities without requiring additional development. This ideal is achieved by a design pattern where community-developed physics "modules" and all MOOSE-based applications are libraries and can be linked under flexible Lesser General Public License (LGPL) version 2.1 licensing, enabling conglomerate applications to be created from existing capability. This concept opens new possibilities for collaborating across development teams to build powerful simulation tools that can be seamlessly combined. This strategy has been a paradigm shift in multiple scientific disciplines [2–8].

Several other open-source finite element method (FEM) frameworks do exist each offering a different set of trade-offs and capabilities to users. Both FEniCS [9] and Firedrake [10] offer compact and concise abstractions for finite element (FE) modeling while utilizing industry standard high-performance solvers from PETSc [11,12]. MFEM [13] and deal.ii [14] both allow for extreme flexibility while leaving developers responsible for building complete C++ applications while using interfaces from one of these respective libraries. Additionally, deal.ii contains one of the largest and most comprehensive set of tutorials of any of these projects. MOOSE-based applications also allow for concise abstraction of the underlying equations while allowing users to use flexible extension points including a full C++ application if the developer sees fit. Additionally, MOOSE contains a comprehensive set of community developed physics modules that can be leveraged to build complex multiphysics simulations without redeveloping equation sets commonly used to model a wide variety of phenomena. Finally, MOOSE focuses on multiphysics by offering several out-of-the-box coupling strategies, tools for coupling to existing applications, and a set of solution transfer capabilities applicable for a wide range of coupling strategies. Each of these will be discussed further in this manuscript. MOOSE offers several unique capabilities, which will be briefly highlighted after a brief introduction.

2. Software description

2.1. Software architecture

MOOSE is designed to facilitate the creation of production FEM tools for running high-fidelity, multiphysics simulations. The software is composed of "systems", each one providing an extension point for defining simulation characteristics. These systems communicate using interfaces, decoupling them to allow for greater code reuse. Each system in MOOSE has a specific C++ base class from which application developers inherit from and extend to perform a desired calculation overriding virtual methods to perform unique calculations needed for that application. This approach has two advantages; the foremost is that the application developers have complete control over their calculation using standard C++, should a need to deviate from the prescribed template arise. Secondly, the inheritance chain can be extended beyond the baseline provided by the framework to provide rich capabilities for downstream applications. MOOSE has approximately 40 systems that may be extended. Individually discussing each of these is beyond the scope of this paper, but the systems that provide the core functionality of the framework can be separated into three groups: PDE related terms, material properties, and in-situ postprocessing.

Two systems for defining PDE terms are the Kernel and BoundaryCondition systems. Kernel objects define volumetric integral terms while BoundaryCondition objects define surface integral terms that arise from the derivation of the FEM weak form [15]. For example, the C++ code snippet in Listing 1 is an example of an implementation of an advection term for a PDE. This example illustrates the Kernel system, which supports automatic differentiation (AD) (see Section 2.2). A single method must be extended, precomputeQpResidual.

Listing 1: Code snippet showing Kernel inheritance for the computation of an advective term within a PDE.

```
template <ComputeStage compute_stage>
ADReal
Advection<compute_stage>::precomputeQpResidual()
{
  return _velocity[_qp] * _grad_u[_qp];
}
```

This is a powerful approach to defining equation terms. By defining one method, this advection operator can be used in 1D, 2D, or 3D. It can utilize a constant velocity, couple to a field computed through solving another PDE, utilize a field computed from another application, or even use a velocity field from an experiment. This Kernel will work on one processor or in parallel on 100,000 without modification. Any application needing this term can reuse this object without needing to recode it. Finally, it can also be tested in a controlled, isolated fashion to ensure correctness.

The Material system allows for the definition of "material properties", often representing coefficients in a PDE. These properties may be nonlinear, and may themselves depend on variables in the PDE. This dependence can be propagated automatically through to the system Jacobian matrix using the AD capability of the framework. Once defined, material properties may be consumed by other systems in the framework through a straightforward producer/consumer model. MOOSE then ensures those properties are computed when needed by those objects. Material properties can couple to and depend on other material properties; inter-dependencies are tracked, and calculations are executed in the correct order. This design decouples physics calculations from the coefficients they require, allowing for properties to be shared and reused within and across applications without the need to edit code objects consuming the property.

The term "in-situ postprocessing", a perceived oxymoron, describes a set of systems used for data computation that is typically performed post-simulation which are computed along with other calculations after, during or in-between solves. Systems in this category include the Postprocessor, VectorPostprocessor, and AuxKernel systems. The Postprocessor and VectorPostprocessor systems compute scalar and vector values, respectively (e.g., total heat flux through a side or the total chemical concentration). These calculations can themselves depend on other variables, material properties, or other postprocessed values. MOOSE contains several utilities to assist in the parallel aggregation, broadcasting, and scattering necessary during the postprocessing stage. The AuxKernel system allows the calculation of "field" or spatially varying data using finite element basis functions. In-situ postprocessing enables calculations to be performed in parallel while running at scale. These values can be fed back into other systems, including the calculation of PDE

terms or material properties. This system can also be used to produce visualization images of field data while the simulation runs.

The system and interface architecture of the framework allows application developers to build simulation tools capable of solving anything from basic single physics problems to extensive, cross-disciplinary multiphysics problems only achievable with multiple development teams. The resulting applications, regardless of complexity, are easily extensible and useful for research teams and analysts alike. MOOSE supports several coupling mechanisms: loose (staggered), tight (iteration), and full (monolithic) [7,16], with several ways to customize the behavior of each type of simulation. Coupled physics are not required to share the same mesh or discretization or even exist on the same spatial or time scales. Picard or fixed point iterations may be employed to converge nonlinearities, while built-in or custom logic may be used to communicate in-memory data among multiple coupled simulations.

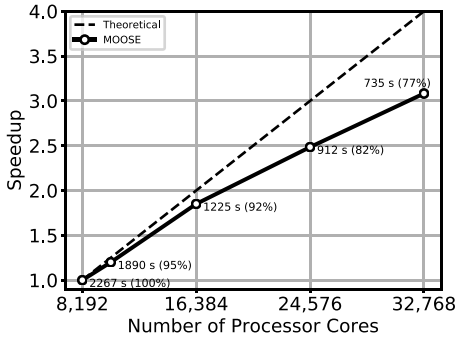

Fig. 1. Scaling study of MOOSE using up to 32,768 processor cores. Each data point is annotated with the total compute time and the parallel efficiency. The total computed time does not include the input/output cost and the mesh preparation.

2.2. Software functionality

2.2.1. Parallelism

MOOSE is designed to support highly parallel calculations involving many physical phenomena. The parallel capability is implemented using a hybrid approach consisting of Message Passing Interface (MPI) calls with optional shared-memory threading available on-node. Traditionally, parallel computing is complicated, involving many low-level serialization, deserialization, and explicit communication routines. MOOSE provides a comprehensive abstraction to parallelism, requiring minimal knowledge (if any) of parallel constructs from application developers. Additionally, the framework supports an advanced mesh "pre-splitting" capability that automatically takes into account necessary "ghosting" needed by contact, mortar, patch recovery, or user-defined, non-local requirements. Pre-split meshes can then be read efficiently in parallel, reducing start-up time and memory use for very large simulations. The pre-split capability has been used to keep per-core memory consumption reasonable (two to three orders of magnitude less depending on the number of processors for billion-element meshes [8]. Scalability of the framework has been demonstrated to over 30,000 processor cores on modern supercomputers for multiple applications including microstructure evolution using the phase-field method [17] and neutron transport [18]. For example, a scaling study is shown in Fig. 1 for a 3D grain growth model based on phase-field equations. The phase-field equations are discretized with 25 grains and 9 order parameters using a first-order Lagrange finite element method on a 3D mesh with 78,643,200 hexahedral elements and 79,300,033 nodes. The resulting system has 713,700,297 unknowns and is solved by the Jacobian-free Newton-Krylov (JFNK) [19] method, employing GMRES [20] and a Schwarz preconditioner [21,22]. The test is conducted using processor counts between 8,192 and 32,768 on the Theta supercomputer at Argonne National Laboratory [23]. Further details of this test can be found in [17,18].

2.2.2. Automatic differentiation (AD)

Efficient solution of nonlinear systems of equations using Newton's method requires accurate computation of the Jacobian matrix or its action on a vector [19]. When this matrix is approximated via matrix-free methods, accurate preconditioning may still be required for efficient linear convergence. Generating an accurate Jacobian matrix or preconditioner can be a difficult and error-prone task. To reduce the burden on application developers, MOOSE has implemented forward-mode AD. Using this capability, application developers only implement residual statements, reducing the amount of code and time needed to implement a new physics object. Sharing physics code among applications can consequently be done with greater confidence. In this way, AD enhances the modularity of MOOSE-based applications.

2.2.3. Sub-applications

The architecture of MOOSE allows separately developed physics simulation tools to be integrated seamlessly via the MultiApp system. As an example, two different research groups might each be focused on different aspects of a problem: one on an engineering scale simulation and one on microstructure evolution. Using MOOSE, both applications can be developed in a uniform, yet flexible way. The two research groups may work independently, focusing on their respective scales. When a higher fidelity, coupled simulation is needed, these two applications can be compiled together, and a multiscale simulation can be performed using the unique MultiApp and Transfer systems without the need to develop additional code to link the applications together [7]. Existing applications not built upon the MOOSE framework may be "wrapped" and linked to MOOSE by implementing a minimal interface, while still supporting either loose or tight coupling strategies and in-memory data transfers [24,25].

2.2.4. Restart

The framework includes a restart system that allows for all aspects of a simulation to be stored in memory or output to checkpoint files. This allows for single applications to be recovered after an unexpected failure (e.g., a power outage), to be restored while conducting iterative solves across multiple applications (see Sub-applications above), as well as to create complex multi-stage initialization for large problems. Notably, this system is extendable so that user-defined data structures can be stored and reloaded when needed in each of these scenarios and when running at scale.

2.2.5. Testing, visualization, and documentation

A modular framework alone is not sufficient to create robust and useful scientific simulation tools. MOOSE also includes infrastructure for testing, visualization, and documentation, which, collectively with the framework, is referred to as the "MOOSE Platform". As with the framework, each of these utilities is designed with similar architectures and to be modular and extendable, allowing application developers to customize these tools. The testing system includes tools for creating unit, regression, and system tests, including tests for error conditions and parallel consistency. The graphical user interface (GUI) is designed to work with derivative applications containing extended objects and custom syntax. The visualization core of the GUI may be

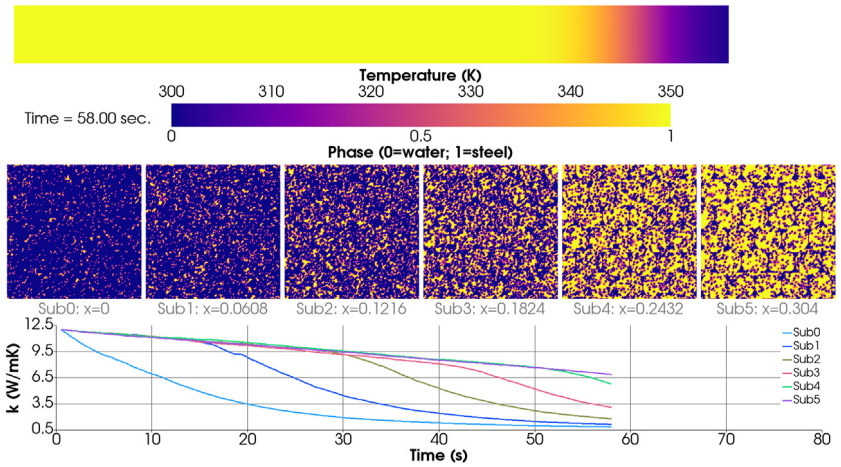

Fig. 2. Results after 58 s of simulation for a thermo-mechanical problem with feedback from a micro-structure calculation that shows the temperature at the engineering scale, the geometry of each degraded micro-structure, and the effective thermal conductivity over time for each of the micro-structure calculations.

scripted to create complex visualizations. Finally, the documentation system operates with single-source markdown files to create websites, presentations, or LaTeX documents and is capable of checking cross-reference consistency as the documentation and source code evolve.

3. Illustrative example

MOOSE is capable of solving complex multiphysics problems, including systems with highly variable space and time scales. For example, the results in Fig. 2 couple an engineering scale problem to a micro-structure scale calculation. The engineering scale physics is porous flow, modeled using Darcy's equation within a cylinder assuming a porous media of closely packed steel spheres, mimicking the experiment detailed in [26]. Each end of the cylinder is attached to a vessel with a prescribed temperature and pressure. The simulation solves the mass, energy, and momentum balance equations for pressure, temperature, and thermally induced strain, respectively using a fully-coupled Newton-based solve. At each time step, a set of micro-structure calculations is performed, resolving the closely-packed spheres. These spheres are degrading, causing an increase in permeability over time. The degradation rate increases with temperature, resulting in a change to the effective thermal conductivity at the engineering scale. Detailed steps for the solve are as follows:

1. At time $t = -1$ the solution for temperature is initialized to 300C, pressure, and displacements are set to zero; the effective thermal conductivity is set to the known value for close-packed spheres of steel in water.
2. The engineering scale problem is solved with a time step of 0.25 s (until $t = 0$). During these steps, the temperature and pressure boundary conditions are increased linearly to 350C and 4000Pa, respectively.
3. The engineering problem computes a solution for a time step, samples the temperature at six evenly spaced points along the domain, and sets the temperature of the corresponding micro-structure calculations.
4. The micro-structure application computes degradation of the steel spheres using the temperature from the engineering scale application and then calculates the effective thermal conductivity based on the changed geometry. The resulting effective thermal conductivity is then applied using linear interpolation to the engineering-scale heat conduction calculation.
5. Steps 3–4 are repeated until a steady-state solution is reached.

4. Impact

Software development is increasingly becoming an integral part of the scientific process. Currently, scientists and engineers develop their own software, spending approximately 30% of their time writing code [27]. These domain scientists often do not follow code reuse best practices [28], which can result in continuous reinvention of software as new researchers inherit opaque works left by a previous generation. By providing a framework that follows best practices for scientific computing, it is possible to aid researchers in combating these shortcomings to improve the quality of scientific research. Developing trust in a framework goes beyond utilizing best practices; it is equally important that the framework itself is built on a foundation of trusted tools. For this reason MOOSE relies on the well-established code bases of libMesh [29] for finite elements and on PETSc [11,12] for leading-edge numerical methods/solvers. Furthermore, MOOSE itself has been subjected to multiple peer-reviews and Software Quality Assurance (SQA) audits.

MOOSE is being used by governments, private industry, and universities both nationally and internationally. For example, MOOSE is used for coupling multiple nuclear energy codes, both external and MOOSE-based [4,24,30]; it is used for modeling porous flow in 3D fractured porous media [31]; and is the basis of multiple efforts to model sub-surface exploration [5,32–34].

5. Conclusions

MOOSE is a software framework that allows scientists to develop state-of-the-art, scalable applications without worrying about parallel, finite element, or solver implementation details. Moreover, the modular design of MOOSE allows these applications to be combined easily, encouraging reuse and simplifying construction of multiscale, multiphysics simulations. The framework has attracted a sizable research community, pushing the frontiers of scientific computing, solving PDE's with tens of billions of unknowns on tens of thousands of processors. MOOSE development continues to focus on robust, high-performance solution algorithms that work across broad scales, from serial to hundreds of thousands of processes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was funded under multiple programs and organizations: The Idaho National Laboratory LDRD program, and The Department of Energy Nuclear Energy Advanced Modeling and Simulation and Consortium, United States of America for Advanced Simulation of Light Water Reactors programs. This research made use of the resources of the High Performance Computing Center at the INL. This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 with the US Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

[1] Gaston D, Newman C, Hansen G, Lebrun-Grandie D. MOOSE: A parallel computational framework for coupled systems of nonlinear equations. Nucl Eng Des 2009;239(10):1768–78.
[2] Keyes DE, McInnes LC, Woodward C, Gropp W, Myra E, Pernice M, et al. Multiphysics simulations: Challenges and opportunities. Int J High Perform Comput Appl 2013;27(1):4–83.
[3] Tonks MR, Gaston D, Millett PC, Andrs D, Talbot P. An object-oriented finite element framework for multiphysics phase field simulations. Comput Mater Sci 2012;51(1):20–9.
[4] Ortensi J, DeHart MD, Gleicher FN, Wang Y, Schunert S, Alberti AL, et al. Full core TREAT kinetics demonstration using Rattlesnake/BISON coupling within MAMMOTH. Technical report, Idaho National Lab.(INL), Idaho Falls, ID (United States); 2015.
[5] Lesueur M, Casadiego MC, Poulet T, Veveakis M. Framework for multi-scale flow simulation of deformable rocks. In: International workshop on bifurcation and degradation in geomaterials. Springer; 2017, p. 475–80.
[6] Nakhmanson S, Mangeri J, Pitike K, Kuna L, Jokisaari A, Alpay S, et al. Ferret: an open-source code for simulating thermodynamical evolution and phase transformations in complex materials systems at mesoscale. In: APS march meeting abstracts; 2017.
[7] Gaston DR, Permann CJ, Peterson JW, Slaughter AE, Andrš D, Wang Y, et al. Physics-based multiscale coupling for full core nuclear reactor simulation. Ann Nucl Energy 2014.
[8] Gaston DR. Parallel, asynchronous ray-tracing for scalable, 3D, full-core method of characteristics neutron transport on unstructured mesh (Ph.D. thesis), Massachusetts Institute of Technology; 2019.
[9] Alnæs MS, Blechta J, Hake J, Johansson A, Kehlet B, Logg A, et al. The fenics project version 1.5. Arch Numer Softw 2015;3(100).
[10] Rathgeber F, Ham DA, Mitchell L, Lange M, Luporini F, Mcrae ATT, et al. Firedrake: Automating the finite element method by composing abstractions. ACM Trans Math Software 2016;43(3):24:1–27.
[11] Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, et al. PETSc users manual. Technical report, ANL-95/11 - Revision 3.11, Argonne National Laboratory; 2019.
[12] Balay S, Gropp WD, McInnes LC, Smith BF. Efficient management of parallelism in object oriented numerical software libraries. In: Arge E, Bruaset AM, Langtangen HP, editors. Modern software tools in scientific computing. Birkhäuser Press; 1997, p. 163–202.
[13] Anderson R, Andrej J, Barker A, Bramwell J, Camier J-S, Cerveny J, et al. MFEM: a modular finite element methods library. 2019, arXiv preprint arXiv:1911.09220.
[14] Bangerth W, Hartmann R, Kanschat G. Deal.II—a general-purpose object-oriented finite element library. ACM Trans Math Softw 2007;33(4):24.
[15] Fish J, Belytschko T. A first course in finite elements. Wiley; 2007.
[16] Novascone S, Spencer B, Andrs D, Williamson R, Hales J, Perez D. Results from tight and loose coupled multiphysics in nuclear fuels performance simulations using BISON. Technical report, Idaho National Laboratory (INL); 2013.
[17] Kong F, Stogner RH, Gaston DR, Peterson JW, Permann CJ, Slaughter AE, et al. A general-purpose hierarchical mesh partitioning method with node balancing strategies for large-scale numerical simulations. In: 2018 IEEE/ACM 9th workshop on latest advances in scalable algorithms for large-scale systems (ScalA). IEEE; 2018, p. 65–72.
[18] Kong F, Wang Y, Schunert S, Peterson JW, Permann CJ, Andrš D, et al. A fully coupled two-level Schwarz preconditioner based on smoothed aggregation for the transient multigroup neutron diffusion equations. Numer Linear Algebra Appl 2018;25(3). e2162.
[19] Knoll DA, Keyes DE. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. J Comput Phys 2004;193(2):357–97.
[20] Saad Y, Schultz MH. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J Sci Stat Comput 1986;7(3):856–69.
[21] Kong F, Cai X-C. A highly scalable multilevel Schwarz method with boundary geometry preserving coarse spaces for 3D elasticity problems on domains with complex geometry. SIAM J Sci Comput 2016;38(2):C73–95.
[22] Smith B, Bjorstad P, Gropp W. Domain decomposition: Parallel multilevel methods for elliptic partial differential equations. Cambridge University Press; 2004.
[23] Argonne Leadership Computing Facility, Argonne National Laboratory.
[24] Martineau R, Andrš D, Carlsen R, Gaston D, Hansel J, Kong F, et al. Multiphysics for nuclear energy applications using a cohesive computational framework. In: Multiphysics for nuclear energy applications using a cohesive computational framework. ANS; 2019.
[25] Novak A, Romano P, Wendt B, Rahaman R, Merzari E, Kerby L, et al. Preliminary coupling of OpenMC and Nek5000 within the MOOSE framework. In: Proceedings of PHYSOR; 2018.
[26] Pamuk MT, Özdemir M. Friction factor, permeability and inertial coefficient of oscillating flow through porous media of packed balls. Exp Therm Fluid Sci 2012;38:134–9.
[27] Wilson G, Aruliah D, Brown CT, Hong NPC, Davis M, Guy RT, et al. Best practices for scientific computing. PLoS Biol 2014;12(1). e1001745.
[28] Morisio M, Ezran M, Tully C. Success and failure factors in software reuse. IEEE Trans Softw Eng 2002;28(4):340–57.
[29] Kirk BS, Peterson JW, Stogner RH, Carey GF. `libMesh`: A C++ library for parallel adaptive mesh refinement/coarsening simulations. Eng Comput 2006;22(3–4):237–54. http://dx.doi.org/10.1007/s00366-006-0049-3.
[30] DeHart MD, Ortensi J, Gleicher FN, Wang Y, Schunert S, Baker BA. Research in support of TREAT kinetics calculations using Rattlesnake/BISON coupling within MAMMOTH. Technical report, Idaho National Lab.(INL), Idaho Falls, ID (United States); 2016.
[31] Schädle P, Zulian P, Vogler D, Bhopalam RS, Nestola MG, Ebigbo A, et al. 3D non-conforming mesh model for flow in fractured porous media using Lagrange multipliers. Comput Geosci 2019.
[32] Ennis-King J, La Force T, Wilkins A. Community code for simulating CO_2 storage: Modelling multiphase flow with coupled geomechanics and geochemistry using the open-source multiphysics framework MOOSE. In: 14th greenhouse gas control technologies conference Melbourne. 2018. p. 21–6.
[33] Poulet T, Schaubs P, Rachakonda PK, Douglas G, Lester D, Metcalfe G, et al. The CSIRO Groundwater Cooling Project — Cooling Australia's Latest Supercomputer with Groundwater.
[34] Regenauer-Lieb K, Poulet T, Veveakis M. Next generation resource discovery linking geophysical sensing, modelling and interpretation. ASEG Ext Abstr 2016;2016(1):1–5.