Continued performance improvement and integration of MOOSE's thermal-hydraulics capabilities

INL/RPT-24-80844
Revision 1

Nuclear Energy Advanced Modeling and Simulation (NEAMS)

M3 Milestone Report

SEPTEMBER 2024

Peter German
Oana Marin
Guillaume L. Giudicelli
Joshua E. Hansel
Alexander D. Lindsay
Daniel C. Yankura
Lise Charlot

Computational Frameworks

Ramiro O. Freile
Mauricio E. Tano

Thermal Fluids Systems Methods and Analysis

Idaho National Laboratory

Battelle Energy Alliance manages INL for the U.S. Department of Energy's Office of Nuclear Energy

INL/RPT-24-80844
Revision 1

Continued performance improvement and integration of MOOSE's thermal-hydraulics capabilities

M3 Milestone Report

Peter German
Oana Marin
Guillaume L. Giudicelli
Joshua E. Hansel
Alexander D. Lindsay
Daniel C. Yankura
Lise Charlot
Computational Frameworks

Ramiro O. Freile
Mauricio E. Tano
Thermal Fluids Systems Methods and Analysis

September 2024

Idaho National Laboratory
Department of 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

ABSTRACT

This work introduces performance, robustness and workflow improvements to Multiphysics Object-Oriented Simulation Environment (MOOSE)-based thermal-hydraulics solvers. It presents work related to the acceleration of segregated fluid dynamics algorithms, which show approximately a factor of 10 speedup compared to the preceding implementation. Additionally, we discuss approaches to use advanced, Schur complement-based, field split preconditioners for monolithic solution algorithms relying on the finite volume method. The presence of the Rhie-Chow interpolation makes the utilization of this preconditioner challenging, but the results indicate that for a moderately large problem a factor of 3.4 speedup can be achieved in conjunction with a factor of 3.5 reduction in memory usage. Furthermore, we introduce several pseudo-time stepping approaches to MOOSE for the robust convergence for cases when steady-state solves don't converge due to the initial guesses being too far from the solution in Newton's method. Every MOOSE-based application has access to this algorithm and can benefit from its use. Moreover, several new avenues have been presented for importing meshes from commercial software which make meshing easier. Lastly, the Component system within the Thermal-Hydraulics Module (THM) of MOOSE is abstracted by separating geometry- and physics-related properties.

SUMMARY

The work in this report covers multiple performance- and robustness-related improvements on Multiphysics Object-Oriented Simulation Environment (MOOSE)-based thermal-hydraulics solvers together with improvements of the meshing workflow and advances on component abstraction. More specifically:

- A new assembly algorithm has been implemented for the acceleration of finite volume-based segregated solution routines. This resulted in an order of magnitude faster simulations, enabling the utilization of the tool for the engineering purposes.
- New preconditioning techniques have been provided for the monolithic, Newton's method-based solution algorithms that rely on the finite volume method. This differs from previous attempts because the effectiveness of the preconditioner changes with the existing pressure diagonal matrix which is the result of the application of the Rhie-Chow interpolation. The suggested Schur complement-based field split preconditioners showcased significant speedup in computation time and reduction in memory consumption.
- A pseudo-time marching algorithm has been added to MOOSE with multiple approaches for time step selection. Based on the first tests, these approaches yield robust, many times better convergence than the existing naive time stepping approaches.
- Several new mesh import paths have been explored using advanced simulation and meshing tools such as StarCCM+®.
- The component system in the thermal-hydraulics module of MOOSE has been changed to split geometry- and physics-related functionalities. This enables a common layer for MOOSE-based system codes.

ACKNOWLEDGMENTS

This report was authored by a contractor of the U.S. Government under Contract DE-AC07-05ID14517. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. Funding was provided by the Nuclear Energy Advanced Modeling and Simulation program.

This research made use of the resources of the High Performance Computing Center at Idaho National Laboratory, which is supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517.

Page intentionally left blank

CONTENTS

FIGURES

TABLES

Page intentionally left blank

ACRONYMS

CFD	Computational Fluid Dynamics
DOE	Department of Energy
FVM	Finite Volume Method
HPDDM	High-Performance unified framework for Domain Decomposition Methods
INL	Idaho National Laboratory
LSC	Least Squares Commutator
MCRE	Molten Chloride Reactor Experiment
MOOSE	Multiphysics Object-Oriented Simulation Environment
NEAMS	Nuclear Energy Advanced Modeling and Simulation
PETSc	Portable Extensible Toolkit for Scientific Computation
SIMPLE	Semi-Implicit Method for Pressure Linked Equations
VTB	Virtual Test Bed

Page intentionally left blank

Continued performance improvement and integration of MOOSE's thermal-hydraulics capabilities

M3 Milestone Report

1. INTRODUCTION

The simulation of fluid flows is a critical component of the design and operation of advanced nuclear reactors. The thermal-hydraulics codes under the umbrella of the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program of the Department of Energy (DOE) provide a wide variety of simulation approaches, ranging from lumped parameter, one-dimensional system-level tools to high-fidelity multi-dimensional Computational Fluid Dynamics (CFD) tools. The main objective of the work presented in this report is to enhance the performance and robustness of the MOOSE-based [7] thermal-hydraulics codes and also to explore pathways to simplify the meshing workflows for faster model generation.

The performance- and robustness-related improvements target critical applications such as the simulation of Molten Salt Reactors (MSRs). They introduce improvements in the system assembly routines of the segregated solution algorithms and the preconditioning of the monolithic solution algorithms. Both improvements pertain to solvers that rely on the Finite Volume Method (FVM) for spatial discretization. The new assembly algorithm for the segregated solver involves enabling the assembly of linear systems directly in MOOSE without relying on any of the machinery required for solving the Navier-Stokes equations using Newton's method. This embraces a new paradigm for MOOSE, which directly assembles system matrices and right hand sides instead of Jacobians and residuals. With this new assembly methodology, the choice of how to linearize a given problem is now given to the application developer. The preconditioning of the monolithic Jacobian resulting from applying Newton's method to the incompressible and weakly-compressible Navier-Stokes equations is a challenging task, considering that the equations pose a saddle point problem. The Schur complement-based field split preconditioning has been investigated for finite element spatial discretization in [1]. However, the Schur complement preconditioners discussed in [1] do not translate well to the finite volume realm due to the Rhie-Chow interpolation [8] that introduces a diagonal block for the pressure variable. For this reason, in this paper, we present additional preconditioning techniques that result in considerable gains in computation time and memory usage compared to direct solution-based algorithms.

To improve the robustness of the thermal-hydraulics solvers, we introduce a pseudo-time continuation algorithm for Newton's method for steady-state simulations [9]. This marches towards the steady-state of the problem using a transient simulation. Typical applications for this algorithm are cases where the initial guess is too far from the solution and Newton's method experiences convergence issues. The key challenge in this scenario is the determination of the pseudo-time step length at each iteration. Several approaches have been implemented including the Switched Evolution Relaxation (SER) and the Residual Difference Method (RDM) algorithms which rely on the evolution of the residuals. This feature has been added directly to the MOOSE framework, therefore it is available to every MOOSE-based application.

The meshing of complex geometries is a significant portion of the model development for nuclear reactors. To accelerate development workflows, we explored several new pathways to import meshes that were generated by advanced CAD-based meshing tools such as StarCCM+[10] and Ansys Meshing[11]. These pathways are important for future developments that target polyhedral element support within MOOSE.

Lastly, we introduce the separation of the geometry- and physics-related functionalities in the Component system of MOOSE's thermal-hydraulics module [12]. This paves the path towards improved code reusability between MOOSE-based system codes.

The report is partitioned as follows: Section 2 discusses the improvement in the assembly routines of the segregated solves; Section 3 presents different preconditioning techniques for the monolithic solution approach; Section 4 showcases the new pseudo-time marching algorithms; while Sections 5 and 6 cover the meshing workflow related work and the component abstraction, respectively.

2. IMPROVEMENT OF SEGREGATED FLUID DYNAMICS SOLVERS IN MOOSE

This section provides a brief review of previous segregated solution methodologies for fluid dynamics problems in MOOSE, discussing both the advantages and limitations with respect to required computational resources. Subsequently, we discuss strategies to enhance solver efficiency, specifically through the optimization of the assembly procedure for individual linear systems. The improvements are demonstrated quantitatively with two case studies:

1. the analysis of a canonical flow within a three-dimensional wavy pipe,
2. the analysis of the flow through a molten salt reactor loop.

2.1. The SIMPLE algorithm in MOOSE

Recent advancements in the MOOSE framework have introduced features that enable the separation of multiple partial differential equations in the same problem into corresponding linear/nonlinear systems instead of including all of them into one, monolithic system. These developments enabled the implementation of conventional, segregated solution algorithms for fluid dynamics problems in the Navier-Stokes module of MOOSE [13, 14]. Currently, the module provides a steady-state fluid dynamics solver using the Semi-Implicit Method for Pressure Linked Equations (SIMPLE) algorithm [15] which is discussed using the notation described in [16] for the Navier-Stokes equations in the following form:

$$\nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = \nabla \cdot \left(\mu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) \right) - \nabla p + \rho \mathbf{g} \,, \tag{1a}$$

$$\nabla \cdot (\rho \mathbf{u}) = 0 \,, \tag{1b}$$

where $\mathbf{u}$ and p are the velocity and the pressure fields, respectively, while μ denotes the dynamic viscosity, ρ the density, and $\mathbf{g}$ the gravitational acceleration vector. We take the momentum equation and linearize it by lagging the advective mass flux. With this, we can express the equation in a semi-discretized form with iteration index n:

$$\mathbf{A}\left(\mathbf{u}^{n-1}\right)\mathbf{u}^{n} + \mathbf{H}\left(\mathbf{u}^{n-1}\right) = -\nabla p \,, \tag{2}$$

where $\mathbf{A}\left(\mathbf{u}^{n-1}\right)$ describes the diagonal terms of the linearized momentum equation computed using the previous velocity iterate $\mathbf{u}^{n-1}$, while $\mathbf{H}\left(\mathbf{u}^{n-1}\right)$ includes the off-diagonal terms multiplied by the previous velocity iterate and every source term besides the pressure gradient on the right-hand side. For the sake of clarity, the notation of the solution dependence of the $\mathbf{A}$ and $\mathbf{H}$ operators is dropped in the subsequent expressions. Next, we multiply Eq. (2) by the density and the inverse of the diagonal matrix $\mathbf{A}$ (which can be computed easily):

$$\rho \mathbf{u}^{n} + \rho \mathbf{A}^{-1}\mathbf{H} = -\rho \mathbf{A}^{-1} \nabla p \,. \tag{3}$$

Then, the continuity constraint ($\nabla \cdot (\rho \mathbf{u}^{n}) = 0$) is enforced to get:

$$\nabla \cdot \left(\rho \mathbf{A}^{-1}\mathbf{H} \right) = -\nabla \cdot \left(\rho \mathbf{A}^{-1} \nabla p \right) \,. \tag{4}$$

With this, we have two equations: an equation for momentum and another equation for pressure. These two equations can be solved in an iterative manner described in Algorithm 1. Based on [15], the algorithm can exhibit convergence issues if the pressure solution update is not relaxed, as in step 4 of the algorithm. The iteration is carried out until we hit a maximum number of iterations ($N_{\max}$) or nonlinear residuals of the equations are below certain tolerances ($||R_{\text{mom}}|| > \tau_{\text{mom}}$ or $||R_{\text{p}}|| > \tau_{\text{p}}$). Although it is not explicitly stated in the algorithm, the momentum equation is relaxed as well to preserve matrix positivity which helps iterative convergence [17].

Algorithm 1. The SIMPLE algorithm [15, 16].

while $||R_{\text{mom}}|| > \tau_{\text{mom}}$ or $||R_{\text{p}}|| > \tau_{\text{p}}$ and $n < N_{\max}$ **do**

1. Solve Eq. (2) to predict a velocity $\mathbf{u}^*$, using p^{n-1} and the previous face mass flux guess $(\rho\mathbf{u})_f^{n-1}$.
2. Solve Eq. (4) for the intermediate pressure guess p^*.
3. Correct the face mass fluxes using the rearranged form of Eq. (3) together with a face interpolation:

$$(\rho\mathbf{u})_f^n = -\left(\mathbf{A}^{-1}\mathbf{H}\right)_f - \left(\mathbf{A}^{-1}\nabla p^*\right)_f , \tag{5}$$

4. Relax the pressure update to get the new pressure guess using $\lambda \in (0,1]$:

$$p^n = \lambda p^* + (1-\lambda)p^{n-1} , \tag{6}$$

end while

2.2. Limitations of the previous implementation

The SIMPLE algorithm has been implemented in the Navier-Stokes module of MOOSE reusing as much of the original system assembly routines as possible [13]. However, the original assembly routines have been tailored towards building finite element Jacobian matrices and residual vectors for Newton's method which incurred a considerable overhead for iterative methods such as SIMPLE which typically require more iterations for convergence. To quantify this overhead, a simple 3D T-junction test case is used. The geometry is presented in Figure 1. The problem is characterized by a Reynolds number of 110, indicating laminar fluid flow. The computational domain consists of 109,375 cells, which corresponds to a system of 437,500 degrees of freedom altogether [1]. We measured the contribution of the existing assembly process to the total runtime

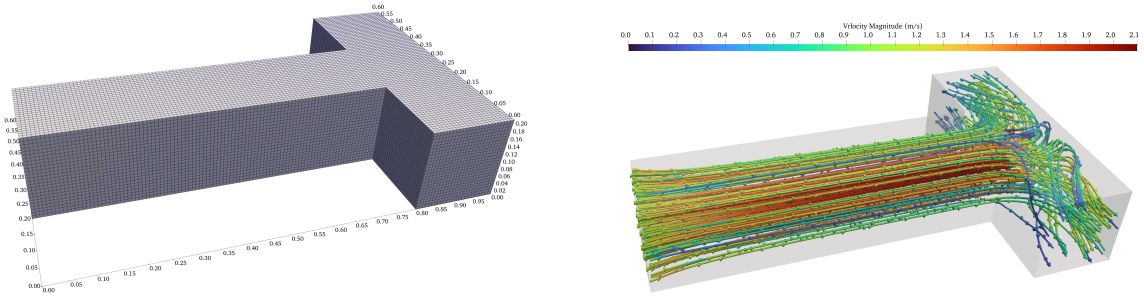

Figure 1. Geometry of the 3D T-junction (left) and the reference velocity solution obtained using the Navier-Stokes module of MOOSE (right) [1].

utilizing Google's gperftools[1] package. The simulations ran for 200 iterations of the SIMPLE algorithm (see Algorithm 1). The effect of changing the number of processors allocated for the simulation on the contribution of the assembly process to the total runtime has also been investigated. The results are presented in Figure 2.

It can be observed that the assembly process takes approximately 80% of the runtime if we use 1–8 processors. Beyond this, due to the increasing parallel communication costs, the contribution of the assembly decreases to 50%. Based on the results presented here, we conclude that the reduction of assembly costs is a priority for reducing the runtime of fluid dynamics simulations that rely on segregated solvers.

[1]https://github.com/gperftools/gperftools (Accessed 08/29/2024)

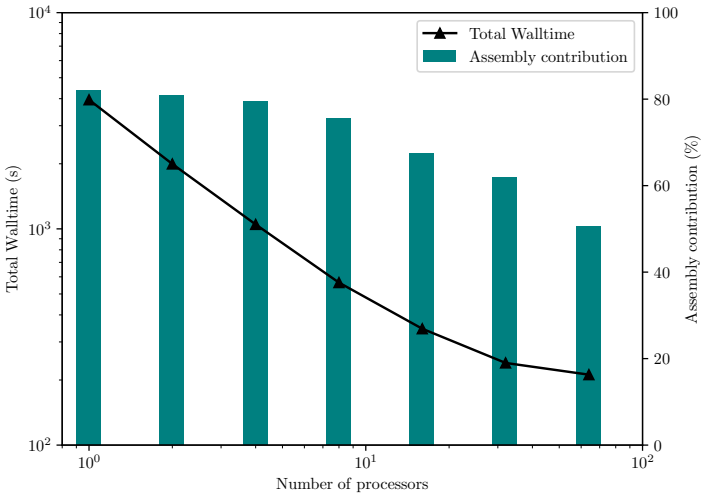

Figure 2. The contribution of the assembly process to the total runtime for the 3D T-junction [1].

2.3. Implementation details for the improved assembly algorithm

To accelerate the assembly process, several new features have been implemented in MOOSE and the Navier-Stokes module:

1. The ability to create purely linear systems within MOOSE without any machinery connecting them to nonlinear solvers based on Newton's method. This was achieved by utilizing the `LinearImplicitSystem` class from `libMesh` [18]. With this, the users have input file level control over the creation of multiple nonlinear and linear systems together in the same problem. This feature is available in every MOOSE-based application, and can be potentially used for segregated solution algorithms in system codes as well.

2. The ability to populate a linear system using FVM. This means that two new assembly loops have been added to populate the system matrix and right hand side: one that loops over the faces to add face flux contributions and another that loops over the elements to add volumetric contributions. These assembly loops utilize user-facing objects called `LinearFVKernels`, `LinearFVBCs` and `LinearFVVariables`.

3. `LinearFVVariables` allow a considerable speedup when it comes to the assembly routine but still behave as traditional variables in MOOSE when it comes to postprocessing and manipulation using the auxiliary system. This ensures that the new finite volume system remains natively coupleable with other moose applications.

4. Considering that the SIMPLE algorithm requires significantly more iterations for convergence, we adopted the following, commonly used strategies for the finite volume systems:

    - We use a system matrix which is as sparse as possible. Each row of the system matrix has one entry for the diagonal, and one entry for each degree of freedom that represents a neighboring cell. This means that every term that would require access to degrees of freedom associated with the other neighbors of the neighbors is treated explicitly. Typical examples include: the nonorthogonal correction term in the discretized form of the diffusion operator, the gradient terms in certain advected quantity interpolation schemes.

- The terms involving gradients are lagged, the gradients are computed using the previous iterate. A typical example is the the nonorthogonal correction term in the discretized form of the diffusion operator. The gradients for the variables are computed in advance using a new loop which is dedicated to computing gradients for every element in one sweep.
- The previous implementation of FVM in MOOSE relied on automatic differentiation [19] for computing the entries of the system matrix (traditionally the Jacobian). The new assembly system does not rely on this feature, considering that it comes with a computational overhead and the carried derivatives would not be utilized.

During the development process of the new capabilities, special attention has been paid to be numerically equivalent to the previous implementation within MOOSE. Within this effort we added a comparison case and additional method of manufactured solutions (MMS)-based verification cases to ensure the code solves the Navier-Stokes equations. These comparison and verification cases are openly available in the test suite[2] of the Navier-Stokes module of MOOSE.

2.4. Results

In this section, we present two three-dimensional models to showcase the improvements in the execution time compared to previous, finite volume-based implementations in MOOSE. Additionally, for comparison, we present execution times compared to other FVM-based open-source and commercial CFD solvers as well.

2.4.1. Flow in a wavy pipe

The geometry and the mesh of the wavy pipe used in this study is presented in Figure 3. The mesh has been prepared using CUBIT [20] based on the extrusion of a two-dimensional unstructured mesh. The mesh consists of approximately 305,000 cells yielding approximately 1.2 million degrees of freedom. We assume an inlet-outlet problem. On one side of the pipe we prescribe a uniform inlet velocity profile with $|\mathbf{u}| = 1.0\ \frac{m}{s}$, whereas on the other side we assume a gauge outlet pressure boundary condition of $p = 0\ Pa$. With $\rho = 1\ \frac{kg}{m^3}$ and $\mu = 0.001\ Pa \cdot s$, we expect a Reynold number of 100, indicating laminar flow regime. The reference solution for these conditions is shown in Figure 4. First, we investigate the gains in execution time from the new assembly algorithm. For this, we run the same problem with the previous and current implementations of the SIMPLE algorithm for 200 iterations. The results shown in Table 1 have been obtained using 48 parallel processes on a AMD EPYC® 9654 CPU. To get a clear picture, two more execution times are included for comparison:

1. The execution time for 200 iterations of the upgraded SIMPLE algorithm, with loading the data for initialization instead of initializing everything. This means that the mesh is split and computed for every processor before the simulation. With this, we get a clear picture on how much of the time spent is actually spent in solving the problem.

2. The execution time for a solve with the monolithic system solved using Newton's method. We run the monolithic solve with a similar relative residual level ($R_{\text{monolithic}} < 2 \times 10^{-4}$) that the SIMPLE algorithm achieves in 200 iterations to get a somewhat fair comparison. We note that the residual definition used for the monolithic and SIMPLE algorithms slightly different, therefore the comparison is not entirely fair in this sense.

[2]https://github.com/idaholab/moose/tree/next/modules/navier_stokes (Accessed 08/29/2024)

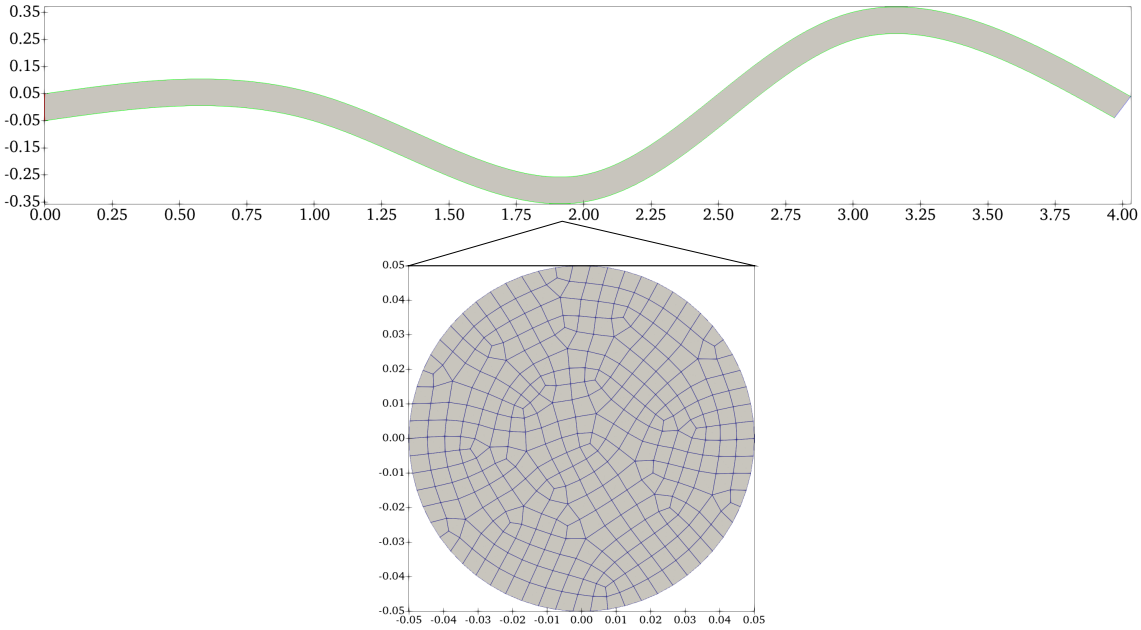

Figure 3. Geometry and mesh of the 3D pipe with dimensions in [m]. (Inlet on the left, outlet on the right)

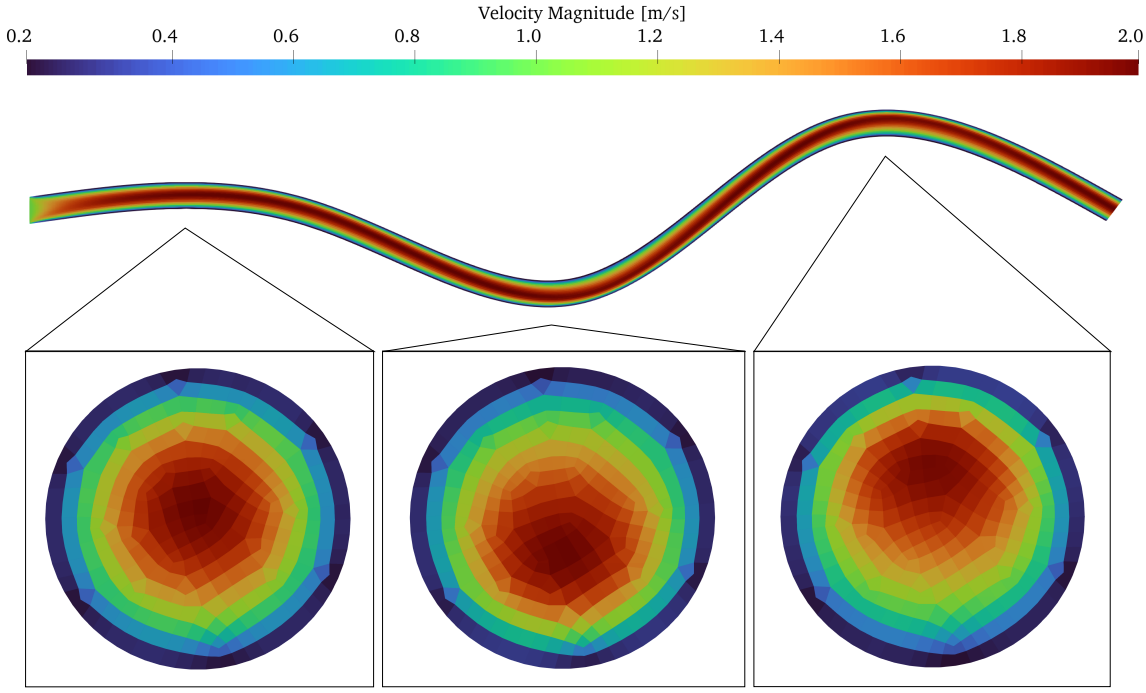

Figure 4. Reference velocity results of the 3D wavy pipe case. The locations of the cuts are $z = 0.6\ m$, $z = 1.95\ m$ and $z = 3.16\ m$.

Based on the results, we conclude that the new assembly algorithm results in a considerable speedup (factor of 6–10) compared to the previous implementation of the SIMPLE algorithm. Furthermore, we also note that the latest segregated solver in MOOSE is more than 45x faster than the monolithic Newton solver, and

Table 1. Comparison of run times and memory usage between different FVM-based solutions algorithms in MOOSE for the 3D wavy pipe case. For the wall time, the achieved speedup factor, while for memory usage, the achieved reduction factor is presented in parentheses.

Solution algorithm	Total wall time (s)	Total memory usage (GB)
Monolithic Newton – LU	1356 (1x)	134 (1x)
SIMPLE with previous assembly	286 (4.7x)	48 (0.36x)
SIMPLE with new assembly	43 (32x)	47 (0.35x)
SIMPLE with new assembly and split mesh	28 (48x)	3.6 (0.03x)

that the segregated solution algorithm requires about a third of the memory used by the monolithic solver. However, for the sake of completeness, we note that the comparison case favors the segregated solver due to:

- The number of cells in the domain is favorable for the segregated solver. It is known that the SIMPLE algorithm needs an increasing number of velocity-pressure iterations with increasing number of cells in the domain[21]. This means that for lower tolerances, considerably more iterations would be needed. The monolithic Newton solver does not suffer from this behavior.
- The monolithic Newton solver uses a direct solver (LU preconditioning) due to the saddle point problem posed by the Navier-Stokes equations. This requires more computation time and memory for the solution of the problem. Recently, a field split preconditioning technique has been developed in MOOSE which utilizes the Schur complement and performs a velocity-pressure iteration in a discrete sense. This has shown considerable reduction in computation time and memory usage for the monolithic solver. However, the current implementation does not support FVM with Rhie-Chow interpolation. For more information, we refer the reader to Section 3.

Following this, we compare the performance of the SIMPLE solver for 200 iterations with the same algorithm in OpenFoam® [16] and StarCCM+® [10], an open-source and commercial tool for solving fluid dynamics problems, respectively. We used 48 parallel processes for every code, however the codes were run on machines with slightly different configurations which introduces some bias into this comparison. The reason behind the utilization of different machines is that not all of the machines are capable of running StarCCM+® in the INL high-performance computing complex. Furthermore, to make the comparison more fair (problem formulation closer to the formulation used in MOOSE) OpenFOAM's `rhoSimpleFoam` solver was used with slight modifications to exclude the energy equation and compressibility effects. Due to this custom version, OpenFOAM® was run using 48 processes on local machine with an AMD Ryzen® Threadripper 3990x chip. The results are summarized in Table 2 together with the used CPU types. The same linear solver tolerances were used with selecting the same types of solvers for the pressure and momentum systems whenever possible. We see that both OpenFOAM® and StarCCM+® are approximately 2–2.5x faster than MOOSE.

Table 2. Comparison of run times and memory usage between different FVM-based software for the 3D wavy pipe test case.

Software	CPU type	Total wall time (s)	Total memory usage (GB)
StarCCM+®	Intel Xeon® Platinum 8268, 2.9 GHz	13	8.7
OpenFOAM®	AMD Ryzen® Threadripper 3990x 2.9 GHz	12	0.9
MOOSE	AMD EPYC® 9654 2.4 GHz	28	3.6

2.4.2. Flow in a molten salt reactor loop

The second example presents a 3D model of a prototype for the Molten Chloride Reactor Experiment (MCRE) built using openly available data[22]. The geometry with its dimensions is depicted in Figure 5. The mesh has been prepared with CUBIT [20] for the purposes of investigating turbulence models in [2]. The geometry is split into approximately 1.57M hexahedral elements. In this work, a laminar flow is assumed with approximately Re=150 in the core cavity. This was achieved by using the nominal density of 3279 $\frac{kg}{m^3}$ but a dynamic viscosity of 1 $Pa \cdot s$ with a volumetric pumping power of 5700 $\frac{N}{m^3}$ within the green block of the mesh embedded in the piping at the top of the reactor. The same problem with the same discretization

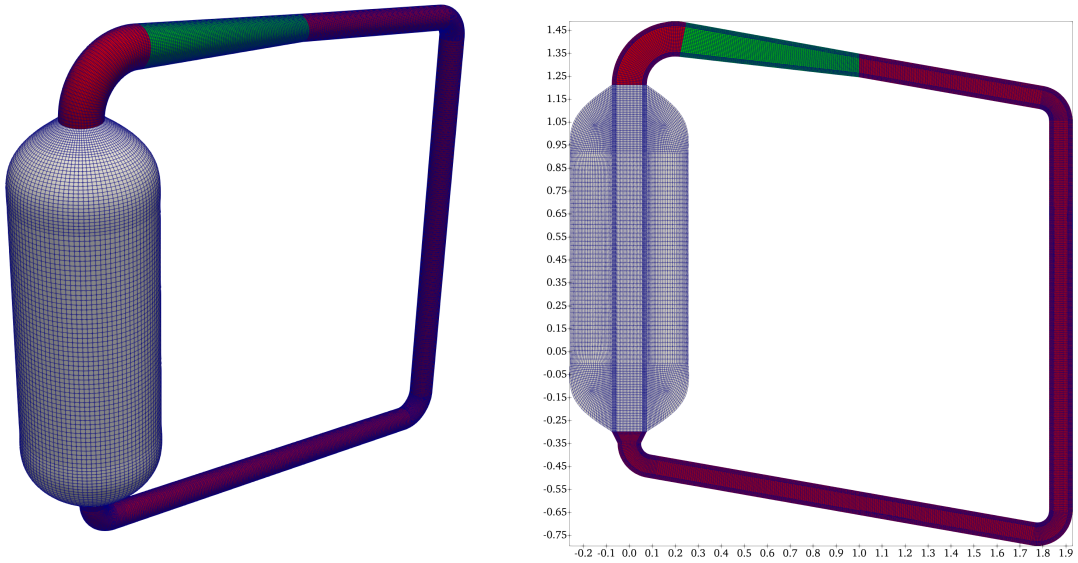

Figure 5. Geometry and mesh of the 3D MCRE-prototype with dimension in [m] [2]. (red: piping, green: pump and heat exchanger, gray: core cavity)

options was run in MOOSE, OpenFOAM® and StarCCM+® for 1000 momentum-pressure iterations. This resulted in residuals in every variable below 1×10^{-4}. The same CPUs were used as for the previous case. The reference results from the three codes are plotted in Figure 6. We see an overall good match in the results from the three codes. The runtime comparison is shown in Table 3. It takes approximately four times longer with MOOSE compared to StarCCM+® . The difference between MOOSE and OpenFOAM® is only a factor of 2x. In terms of memory utilization, we see that both MOOSE and OpenFOAM® use less memory than StarCCM+® .

Table 3. Comparison of run times and memory usage between different FVM-based software for the 3D MCRE-prototype.

Software	Total wall time (min)	Total memory usage (GB)
StarCCM+®	3	31
OpenFOAM®	6	2.6
MOOSE	12	15

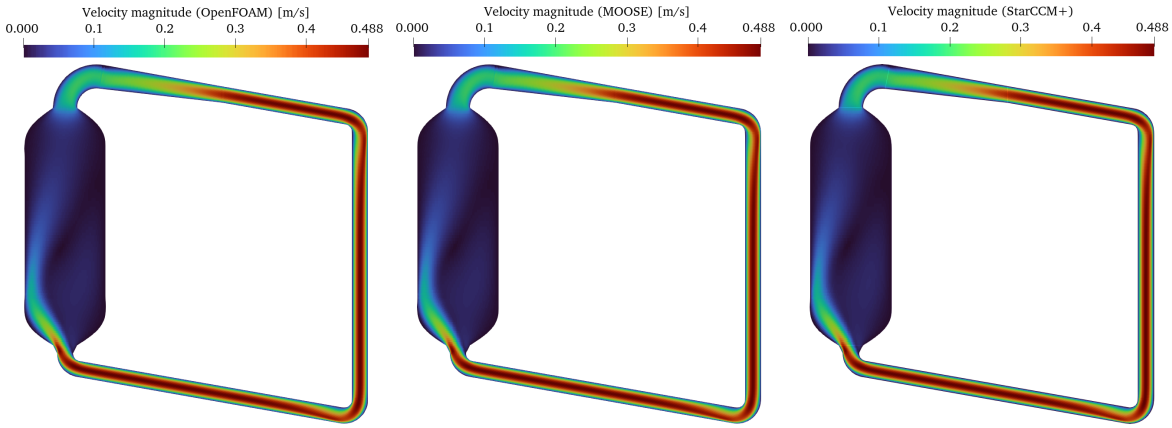

Figure 6. Velocity magnitude at the middle plane of a MCRE-prototype from different solvers [2]. (left - OpenFOAM® , middle - MOOSE, right - StarCCM+®)

2.5. Future work

We see that even though there is significant progress regarding the performance of the segregated fluid dynamics solvers in MOOSE, the runtimes are still longer than those experienced with the other tools in this work. Nevertheless, a MOOSE-native fluid dynamics solver is directly coupleable with other MOOSE-based applications for multiphysics modeling of nuclear systems. The runtimes are now comparable to other CFD tools, making the solver usable for engineering applications. Furthermore, based on preliminary profiling, the following improvements would bring the performance of the MOOSE-based solvers even closer to industry standard:

- The assembly of the momentum system is still relatively expensive, especially for simulations in 3D. It contributes to approximately 24% of the runtime in the case of the molten salt reactor loop example. The reason behind this is that we assemble system matrices for all momentum components separately. Industry codes, on the other hand, utilize the fact that the bulk of the system matrix is the same for all components. Therefore, a considerable speedup could be achieved if the differing parts (such as boundary condition contributions) could be stored and applied right before solve time. This could also reduce the memory footprint, considering that only one matrix would be stored.
- The computation of cell gradients takes up a significant portion of the runtime (approximately 8%). Partially because the assembly loops are reinitialized and operate on local data which is allocated before the assembly. The allocation costs for this local data can be saved if the loops are reinitialized only when the mesh changes so that the local storage is reused every time we assemble the same system. An alternative approach could be to save the gradient operator in a sparse matrix and compute gradients using a simple matrix-vector multiplication.
- The face fluxes are stored in a map-based functor object, which incurs overhead when the values are retrieved. This could be simplified if the functor's underlying storage is replaced with a container with faster access.

Lastly, we note that currently MOOSE only supports the SIMPLE algorithm which is a steady-state solver. Future work involves the extension of this to transient simulations by combining it with PISO [23].

3. FIELD SPLIT PRECONDITIONING FOR FVM-BASED FLUID DYNAMICS SYSTEMS WITH RHIE-CHOW INTERPOLATION

One of the major computational bottlenecks encountered for the monolithic, FVM-based fluid dynamics solver in MOOSE is the preconditioning of the Jacobian matrix. In essence, the challenge is posed by the Navier-Stokes equations presented in Eqs. (1a)–(1b) yielding a saddle point problem. The resulting monolithic matrix is not diagonally dominant meaning that users are limited to the use of direct solution algorithms (e.g. LU-decomposition) whose solve time and memory requirement rapidly scale with the number of unknowns in the problem. This challenge has been addressed in the fluid dynamics community by the utilization of Schur complement-based preconditioning techniques which require the splitting of the Jacobian matrix and residual vector into variable-wise blocks [24–26]. MOOSE utilizes the field split preconditioning interface provided by Portable Extensible Toolkit for Scientific Computation (PETSc) [27] for this purpose, and it has been demonstrated in [1] that it yields a significant reduction in memory costs and runtime for finite element problems. In this work, we expand on the existing capabilities to provide efficient and scalable preconditioning options for the monolithic, FVM-based nonlinear systems.

3.1. Schur complement-based preconditioning

First, we briefly review our solution algorithm using Newton's method with Schur complement-based preconditioning for the linearized Newton iteration. We start from the standard formulation of Newton's method with $J(\mathbf{x})$ denoting the Jacobian matrix, and $R(\mathbf{x})$ being the residual vector:

$$J(\mathbf{x})d\mathbf{x} = -R(\mathbf{x}). \tag{7}$$

We can split this discretized system into variable-wise blocks:

$$\begin{pmatrix} A & G \\ D & C \end{pmatrix} \begin{pmatrix} d\mathbf{u} \\ dp \end{pmatrix} = - \begin{pmatrix} \mathbf{R}_u \\ R_p \end{pmatrix}, \tag{8}$$

where A denotes the momentum diagonal matrix with the discretized momentum advection and the stress tensor operators, while G and D denote the discretized gradient and divergence operators, respectively. Block matrix C is 0 in the traditional incompressible formulation, but depending on the spatial discretization and stabilization techniques, it may hold nonzero entries. In case of FVM, due to the Rhie-Chow interpolation method [8], it does contain nonzero entries. In PETSc, this matrix is factorized into the following, simpler matrices [28]:

$$\begin{pmatrix} I & 0 \\ DA^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & A^{-1}G \\ 0 & I \end{pmatrix} \begin{pmatrix} d\mathbf{u} \\ dp \end{pmatrix} = - \begin{pmatrix} \mathbf{R}_u \\ R_p \end{pmatrix}, \tag{9}$$

where $S = C - DA^{-1}G$ is the Schur complement. Naturally, the ideal preconditioner would be the inverse of the factorized matrix in a form of:

$$\begin{pmatrix} I & -A^{-1}G \\ 0 & I \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -DA^{-1} & I \end{pmatrix}. \tag{10}$$

The solution algorithm using this preconditioner is discussed in Algorithm 2 in detail. We see that this preconditioner involves two solves with matrix A and one solve with matrix S, all requiring their own preconditioners. Additionally, we note that there is another linear solve with A within the Schur complement

every time it is applied to a vector. Preconditioning S is a challenging task, considering it is a dense matrix and can be expensive to compute. In most cases, the Schur complement is not stored explicitly, only its action and inverse action are used through the factorized forms in Eqs. (9) and (10). For Stokes problem, it is known that the pressure mass matrix is spectrally equivalent to the Schur complement and thus can be used as an efficient preconditioner, but for Navier-Stokes problem, more advanced techniques, such as the Least Squares Commutator (LSC) [26] have also been studied. The main advantages of this algorithm over the direct solution techniques include:

- It has a smaller memory footprint, considering that we don't need to store the dense matrices associated with an LU decomposition. Even though the Schur complement is a dense matrix, it is not stored explicitly.
- The solution algorithm scales better than direct solvers, leading to shorter runtimes for large problems.

Algorithm 2. The Schur complement-based field split preconditioning algorithm.

Assume initial conditions: $\mathbf{u}^0,\ p^0$
while $|\mathbf{R}| > \tau$ or $n < N_{\max}$ **do**

1. Compute the following Jacobian and rh.s. blocks using the current solution: $A, G, D, C, \mathbf{R}_u, R_p$
2. Compute an intermediate source for the pressure equation:

$$r_p^* = -R_p + DA^{-1}\mathbf{R}_u. \tag{11}$$

3. Solve the pressure update equation using the Schur complement:

$$dp = S^{-1}r_p^*. \tag{12}$$

4. Solve for the velocity update using the known pressure update:

$$d\mathbf{u} = A^{-1}\left(\mathbf{R}_u - Gdp\right) \tag{13}$$

5. Update the velocity and pressure solutions:

$$\begin{aligned} \mathbf{u}^{n+1} &= \mathbf{u}^n + d\mathbf{u} \\ p^{n+1} &= p^n + dp \end{aligned} \tag{14}$$

end while

3.2. Application of field split capabilities to finite volume problems

In this work, we investigate if the existing Schur complement-based preconditioning techniques in MOOSE are applicable to finite volume problems. To showcase the usability of current options, we select a case involving flow in a T-junction. We assume a uniform inlet velocity of 1 $\frac{m}{s}$, fluid density of $\rho = 1\ \frac{kg}{m^3}$, and dynamic viscosity of $\mu = 0.01\ Pa \cdot s$. These result in an approximate Reynolds number of 100. The geometry of the T-junction is depicted in Figure 7. We prepared two meshes containing 10^4 and 10^5 cells, respectively.

As a first attempt, we use an average interpolation method to determine the advecting velocity at the faces of the control volumes. This ensures that the C matrix block is 0, bringing it closer to the finite element realm where the preconditioning of the Schur complement is better understood. Table 4 summarizes

runtimes and memory usage for different solver settings using 8 processes on a MacBook Pro 2021. For the field split preconditioning, we utilized an inexact LU preconditioner based on low-rank compression for the velocity block [29] and a LSC preconditioner for the Schur complement. BoomerAMG [30] was used as a preconditioner for the Laplacian operators within LSC. We see that the simple LU preconditioner on the

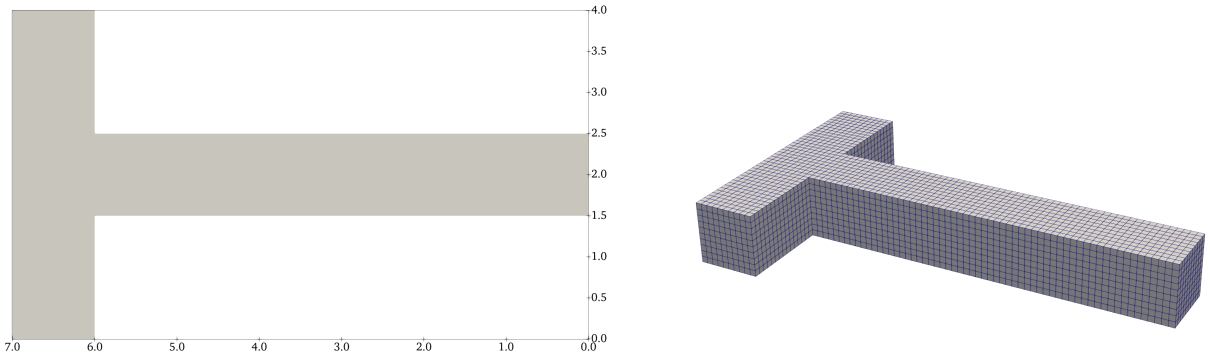

Figure 7. Geometry (left) and mesh (right) of the 3D T-junction with dimensions in [m].

monolithic system outperforms the Schur complement-based field split preconditioner for the case with lower number of cells. However, with the increasing number of cells, the difference in scaling between the two algorithms becomes notable with the field split preconditioner resulting in simulations approximately 11 times faster than the simple LU preconditioner. Furthermore, we also note that the field split preconditioner requires approximately one quarter of the memory needed for a run with the LU preconditioner. However, with a

Table 4. Comparison of run times and memory usage between different preconditioning algorithms in MOOSE for the 3D T-junction using weighted average face velocity interpolation.

Number of cells	Preconditioning	Total runtime (s)	Total memory usage (MB)
10^4	Monolithic LU	14	966
10^4	Field split	18	1,104
10^5	Monolithic LU	1269	20,800
10^5	Field split	114	5,600

weighted average interpolation method for the face velocities, the cell-centered finite volume method applied to the Navier-Stokes equations can exhibit numerical checkerboarding in the pressure field [8]. This effect is presented in Figure 8 comparing the results with a pressure field obtained by employing the Rhie-Chow interpolation.

This numerical checkerboarding is an undesirable effect, therefore we investigate if the field split preconditioner can be used in conjunction with the Rhie-Chow interpolation. This is a more challenging problem because matrix C in Eq. (8) is not 0. In this scenario, the preconditioning of the Schur complement could not be achieved by LSC. For this reason a Successive Over-Relaxation (SOR) preconditioner was created using matrix C. This approach successfully resulted in converging simulations, even though the convergence rate was considerably degraded compared to the case with weighted average interpolation. Furthermore, a preconditioner based on High-Performance unified framework for Domain Decomposition Methods (HPDDM) [24, 31] was also succesfully applied to the Schur complement. The comparison of runtimes and memory usage for the different preconditioning techniques is presented in Table 5. We see that similarly to the weighted average interpolation case, the simple monolithic LU preconditioning outperforms the field split-based techniques for the coarse mesh. However, as we increase the problem size the field split techniques result in a

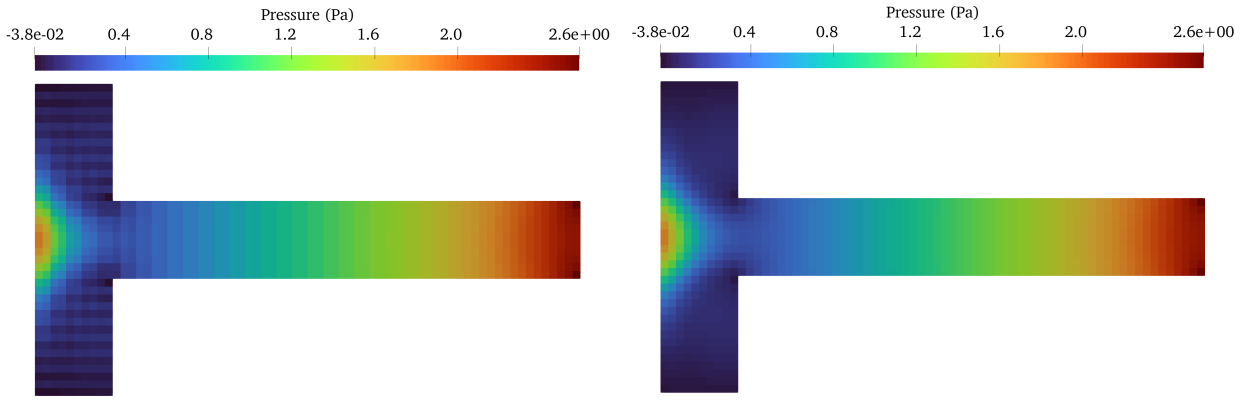

Figure 8. Pressure field in the T-junction with weighted average (left) and Rhie-Chow interpolation (right) for the advecting face velocities.

considerable decrease in computation time and memory usage. The HPDDM-based preconditioner seems to result in slightly faster simulations.

Table 5. Comparison of run times and memory usage between different preconditioning algorithms in MOOSE for the 3D T-junction using Rhie-Chow face velocity interpolation.

Number of cells	Preconditioning	Total runtime (s)	Total memory usage (MB)
10^4	Monolithic LU	20	1,040
10^4	Field split (HPDDM)	55	1,136
10^4	Field split (using C with SOR)	48	1,104
10^5	Monolithic LU	2,776	23,200
10^5	Field split (HPDDM)	767	6,560
10^5	Field split (using C with SOR)	800	6,400

3.3. Future work

Even though it was demonstrated that the Schur complement-based field split preconditioning can be used for systems using FVM and Newton's method, the scalability of the method with cases utilizing the Rhie-Chow interpolation can still be improved. We propose two directions for improvement:

- The utilization of more advanced preconditioners based on the literature, such as the one recently developed and discussed in [25] for a grad-div stabilized equal order finite element discretization (resulting in a non-zero C block):

$$P = \frac{1}{\gamma + \mu} M + C\,, \tag{15}$$

  where M is the pressure mass matrix and γ is the grad-div stabilization parameter. For stable finite element pairs ($C = 0$), grad-div or augmented Lagrange stabilization is typically used to force the Schur complement to become spectrally equivalent to the pressure mass matrix as for Stokes flow. In the future, we may explore addition of augmented Lagrange terms (like is done in [32]) to our finite volume discretization such that Eq. (15) may be an effective preconditioner for our Schur complement.

- The fine tuning of the HPDDM algorithm for the preconditioning of the full Schur complement. The multilevel domain decomposition in HPDDM has many tunable parameters which could be used to enhance its performance for a given problem.

4. IMPROVEMENT OF SOLVER ROBUSTNESS

Solver robustness is defined as the ability for the solver to yield the correct solution in spite of various general difficulties such as:

- The initial guess may be poor and far from the solution causing convergence issues with Newton's method.
- Time step length may be ill-chosen and violate the Courant-Friedrich-Levy conditions for the discretization scheme.
- A combination of material properties resulting in conditions where the given preconditioning techniques don't work.

Several mechanisms have been implemented in MOOSE to increase the solver robustness over the years. The most commonly used in transient simulations is simply to recover from a failed time step solve by cutting the time step and trying to solve again. The cut in the time step results in larger diagonal term, which leads to better-conditioned systems.

The resiliency of MOOSE based thermal-hydraulics and fluid dynamics codes is a priority to the mission of the NEAMS program. Several approaches were suggested, including different nonlinear solver implementations, optimal time stepping strategies, transient viscosity ramp down, and adaptive convergence assessments. This work discusses optimal time stepping strategies developed for pseudo transients to reach steady state for fluid dynamics problems.

4.1. Approaches for time step selection

The pseudo-transient continuation algorithm for Newton's method, based on [9], is presented in Algorithm 3. The key aspect of the transient continuation scheme is the computation of a pseudo time step which is

Algorithm 3. Pseudo-transient continuation with Newton's method

Assume initial conditions: $\mathbf{x}^0$, Δt_0.
while $|\mathbf{R}_{\text{steady}}| > \tau$ or $n < N_{\max}$ **do**
1. Compute the steady-state Jacobian and residual objects: $J(\mathbf{x}^{n-1})$, $R(\mathbf{x}^{n-1})$.
2. compute a mass matrix: M.
3. Solve the following system for the solution update: $d\mathbf{x} = -\left(\frac{1}{\Delta t_{n-1}} M + J(\mathbf{x}^{n-1})\right)^{-1} R(\mathbf{x}^{n-1})$.
4. Update the solution: $\mathbf{x}^n = \mathbf{x}^{n-1} + d\mathbf{x}$.
5. Compute the new residual $R(\mathbf{x}^n)$ and recompute Δt_n.

end while

optimal in a sense that it results in robust transients marching towards steady-state, given that the initial time step is small enough. The change in time step is determined by the steady-state residual behavior from one iteration to another, i.e., a decrease in residual indicates larger time steps are allowed. In contrast, significant increase in the residual indicates a time step decrease is necessary. Following [33], three time step selection approaches have been implemented: Switched Evolution Relaxation, Residual Difference Method, and the Exponential Progression methods. All methods require a parameter α, which controls how sensitive the time step length is with respect to residual changes. The user is also required to supply the initial time step length

Δt_0. Specific choices for α and Δt_0 for fluid dynamics problems are available in [33] or [34]. For the above mentioned methods, the time step in the next iteration can be selected using the following strategies:

1. Switched Evolution Relaxation (SER):

$$\Delta t_k = \Delta t_{k-1} \cdot \left(\frac{R_{k-\ell}}{R_k} \right)^{\alpha}, \tag{16}$$

   where α is a user chosen parameter, R_k is the L^2-norm of the steady-state residual at step k, and the residual at ℓ iterations before is denoted as $R_{k-\ell}$.

2. Residual Difference Method (RDM):

$$\Delta t_k = \Delta t_{k-1} \cdot \alpha^{\frac{R_{k-1}-R_k}{R_{k-1}}}, \tag{17}$$

3. Exponential progression (EXP):

$$\Delta t_k = \Delta t_0 \cdot \alpha^k, \tag{18}$$

Based on the discussion in [33], the Exponential progression method has an infinite growth, therefore specifying a maximum time step length is recommended. Alternatively the user can utilize the automatic steady-state detection in MOOSE.

4.2. Results

We present results for pseudo-transients with different time step selection algorithms for a 2D-RZ model of the Molten Salt Fast Reactors (MSFR) [35] from the Virtual Test Bed (VTB) [36]. We use pseudo transients to obtain a steady-state solution because the nonlinear solver is unable to solve the steady-state problem, despite our best efforts, with uniform initial fields. We only solve for pressure and velocity without considering temperature. We use an initial time step size of $\Delta t_0 = 1\ s$. The first time step is a very difficult solve, with close to 20 nonlinear iterations required to reach convergence for a traditional Newton solve. 20 is chosen as the maximum amount of nonlinear iterations allowed before re-trying the time step with traditional iteration-based time steppers. Every failed time step will therefore incur a 20 nonlinear iteration cost if the residual is stagnating. For the pseudo-time steppers, only one Newton solve is used per time step, as described in Algorithm 3. The convergence of the solution is determined using the following condition:

$$||\mathbf{x}^n - \mathbf{x}^{n-1}|| < 10^{-7} \cdot ||\mathbf{x}^n||\,. \tag{19}$$

The results presented in Table 6 have been obtained using 8 parallel processes a Macbook Pro M1 Max 2021. `IterationAdaptiveDT` results use first the default settings, where the time step is grown by a factor of 2 as long as it succeeds. We also include results obtained by increasing the grow factor to 3 and 4. We see that using a growth factor of 2 results in the most robust and fastest convergence with 0 failed time steps and 43 nonlinear iterations altogether. These values are used as basis for comparison with the pesudo-time stepping algorithms. We carried out the numerical experiments with two different values of the α parameter (1.5 and 2.0) for all three approaches. Based on the results, we conclude that all but one pseudo-transient simulations yield better results than those obtained with `IterationAdaptiveDT`. The best results were obtained using the residual difference method, saving almost 50% in computation time. The exponential progression method, however, did not converge with $\alpha = 2$ with an initial time step size of 1 s. Further investigation revealed that it converges with $\Delta t_0 = 0.5\ s$ in approximately 12 s. This is not surprising, considering that the exponential

Table 6. Comparison of pseudo transient time stepping strategies for the 2D MSFR model with viscosity ramping.

Strategy	Total number of time steps	Number of failed time steps	Total number of nonlinear iterations	Solve time (s)
Constant $\Delta t = 1\ s$	35	3	194	95
Iteration adaptive - factor 2x	5	0	43	21
Iteration adaptive - factor 3x	7	0	57	27
Iteration adaptive - factor 4x	8	1	74	38
SER α=1.5	14	0	14	11
SER α=2	23	0	23	17
RDM α=1.5	16	0	16	12
RDM α=2	13	0	13	10
EXP α=1.5	13	0	13	10
EXP α=2	-	-	-	-

progression method doesn't use residual information but provides a monotonic increase in the time step length. We note an important feature of the pseudo-time transients: unlike other adaptive time steppers in MOOSE, it does not restart the simulation from a previous step if the solve fails. This means that for diverging cases, one is recommended to decrease the initial time step size.

The march to steady-state for the 2D MSFR model involves an increased viscosity at the beginning which is continuously decreased throughout the simulation to help convergence. As a next step, we investigate if removing the viscosity ramp-down strategy influences the convergence results. For this, we start the simulation from $\Delta t_0 = 0.5\ s$. The results obtained with different time steppers are summarized in Table 7. We see that SER yields similar results to the default iteration-based time stepper. On the other hand the RDM-based algorithm yields considerably faster convergence. The exponential progression-based method does not converge with this initial time step size.

Table 7. Comparison of pseudo transient time stepping strategies for the 2D MSFR model without viscosity ramping.

Strategy	Total number of time steps	Number of failed time steps	Total number of nonlinear iterations	Solve time (s)
Constant $\Delta t = 0.5\ s$	59	3	246	125
Iteration adaptive - factor 2x	10	0	60	29
SER α=1.5	42	0	42	30
RDM α=1.5	28	0	28	20
EXP α=1.5	—	—	—	—

5. IMPROVEMENT OF MESHING WORKFLOW

Generating high quality meshes for fluid dynamics problems is a critical step of the simulation workflow. Low quality meshes can result in convergence issues or incorrect results. Therefore, a large portion of the model development phase is spent creating adequate meshes. MOOSE supports several workflows for meshing geometries for various nuclear reactor simulations. It provides native mesh generators for simple problems together with import capabilities for complex meshes generated by third-party software. The focus of this work is to explore robust, easy-to-use meshing workflows for three-dimensional fluid dynamics simulations in MOOSE. The main motivation behind this effort is to minimize the time spent in the meshing phase of the model development. Primarily, we focus on importing meshes from advanced tools with automatic mesh generation options such as StarCCM+® [10].

5.1. Native meshing capabilities in MOOSE

Meshing capabilities in MOOSE have grown enormously over the last few years, driven by the expansion of the reactor module [37]. In-MOOSE meshing considerably facilitates optimization or uncertainty quantification studies with regards to the geometry. The ability to generate mesh on-the-fly with perturbations to the dimensions is a key asset. The reactor module is focused on meshing advanced reactors with regular lattice-based geometries, such as hexagonal lattices in sodium fast reactors and several micro-reactor designs. For these reactors, the reactor module can generate meshes for pins, channels, assemblies, and the core, as well as simple shapes for the top and bottom plenum. These meshes can be used for homogenized coarse-mesh thermal hydraulics, as was done in [38]. In those simulations, the assemblies are homogenized in a few mesh cells, with a porous medium containing the fuel, cladding and coolant. The friction coefficients are set using correlations, which seek to reproduce the pressure drop across the real assembly. If ducts are present in each assembly, then a high friction factor in the transverse direction can limit the cross-flow. Figure 9 shows two examples for meshes generated by MOOSE for coarse mesh thermal-hydraulics simulations.

The capabilities of the reactor module can also be used for simple 2D shapes, for example 2D-RZ models of pebble bed reactors with conical expansion regions. Vertical or angled planes can be defined using parsed curve equations, and the region in between two curves can be filled with quadrilateral elements. The meshed regions can be stitched together on the curves previously specified. MOOSE can currently only mesh using quadrilateral elements in a limited number of configurations. For example, it can fill between the area between two curves with the same number of nodes with quadrilateral elements. However, it can fill an arbitrary region inside a single curve using triangle elements. These limitations make the native meshing of general 2D geometries a lengthy and careful process. The limitations are even more apparent when it comes to the meshing of general geometries in 3D. However, we note that MOOSE also contains several native mesh generators and manipulators routines that can modify and analyze meshes provided by third-party software.

5.2. Importing meshes generated by third-party software

For more complex geometries, especially in 3D, the users can import meshes from external, third-party simulation and meshing tools. MOOSE supports importing a variety of mesh file formats, all of which are specified on MOOSE's website [39]. This capability enhances user flexibility, allowing them to select the most appropriate mesh tailored for their specific application needs. Currently, MOOSE solvers support hexahedral, pyramidal, prismatic, and tetrahedral elements. This work focuses on exploring pathways to import meshes from software products that are easy to use.

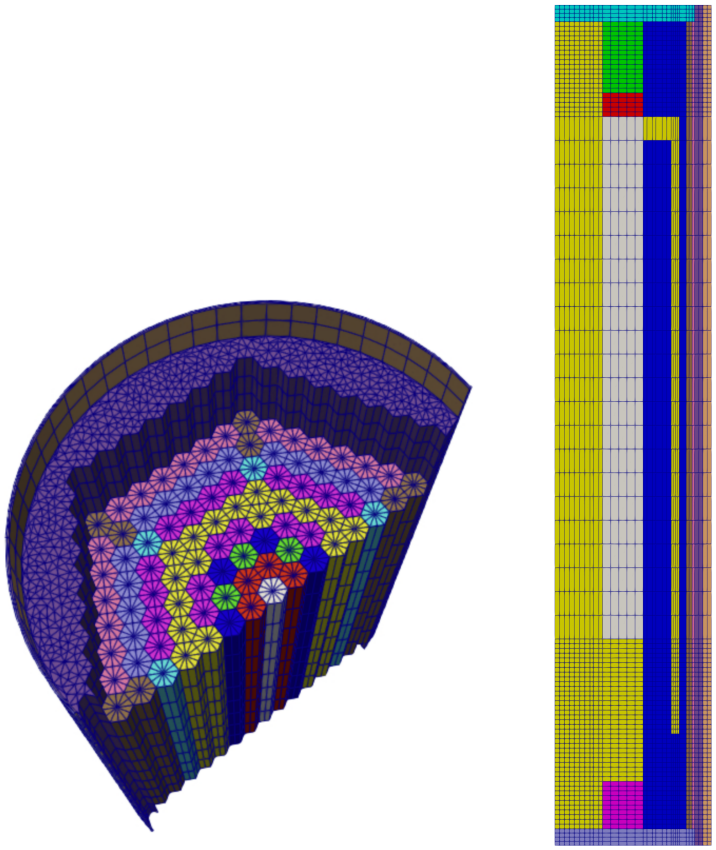

Figure 9. Examples of meshes of reactors generated using MOOSE. (left: High-Temperature Test Facility (HTTF) [3], right: 2D-RZ model of a Pebble Bed Modular Reactor 400 (PBMR-400) [4]).

A summary of the main meshing workflows for some of the most popular third-party meshing tools is shown in Figure 10. The workflows begin with a CAD geometry file that is imported into the different meshing software. Three of the five meshing software products presented in the figure can directly export mesh files in formats supported by MOOSE. SALOME open-source platform [40], Gmsh open-source mesh generator [41], and Cubit [20] can export the generated computational meshes in UNV ('.unv') and EXODUS ('.e') files, respectively. These formats are supported by MOOSE, eliminating the need for any additional file conversion. However, for popular CFD codes with advanced meshing capabilities such as StarCMM+ and ANSYS Meshing [11], the workflow requires and additional step. StarCMM+ provides the capability of generating high quality swept hexahedral meshes through its `Directed Mesh` meshing tool. The generated mesh is first exported from StarCMM+ in CGNS ('.cgns') format and imported into ParaView, an open-source visualization software. From ParaView, the mesh can be exported in Exodus ('.e') format, which is then compatible for direct use in MOOSE. ANSYS meshing software can produce hexahedral meshes in the MSH ('.msh') format. However, the Fluent-generated .msh files are not directly readable by MOOSE. To address this, Cubit can be used to convert the Fluent mesh to Exodus ('.e') format, which is compatible with MOOSE.

Figure 11 presents a 3D mesh of a swirling curved pipe generated using StarCCM+. Using the directed mesh scheme, the 2D inlet surface of the pipe is divided into five blocks: a central square and four evenly distributed circular sections. Next, the edges defining the five blocks are subdivided to create rectangular elements. Finally, the inlet surface is swept from the inlet to the outlet of the geometry, with axial divisions specified in the sweep direction to generate hexahedral elements. The mesh is exported in CGNS format, read into Paraview and ultimately exported in Exodus format.

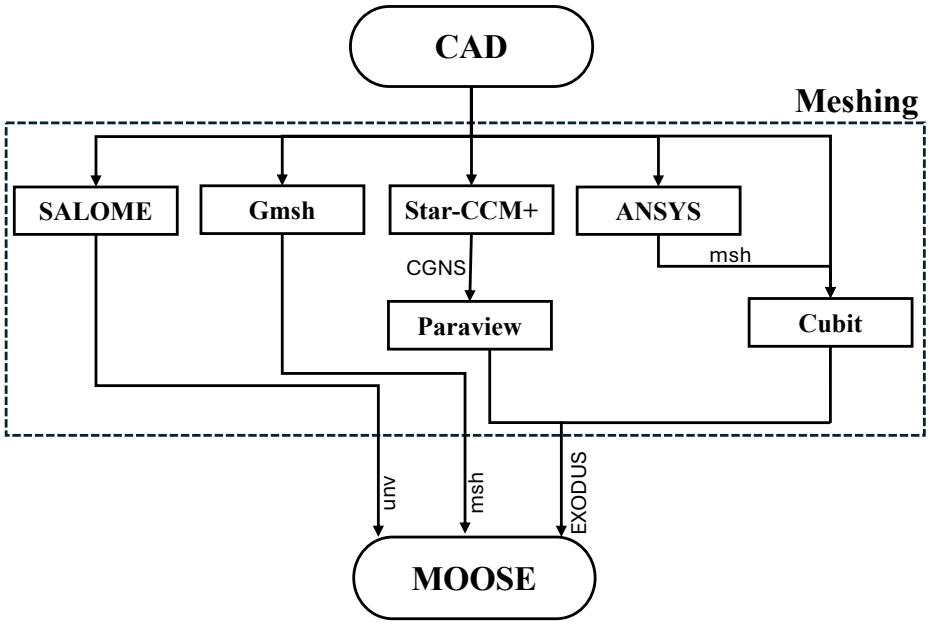

Figure 10. Summary of currently supported third-party software meshing workflows.

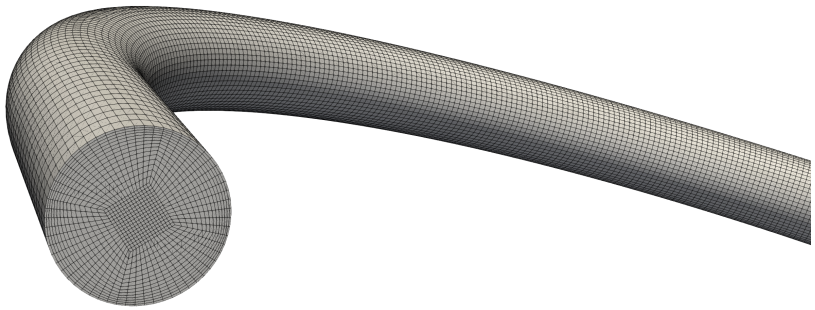

Figure 11. StarCMM+-generated mesh using the Directed Mesh scheme of a swirling curved pipe imported into MOOSE [2, 5].

Figure 12 presents cross sections of 3D meshes of the open source specifications of the Molten Chloride Reactor Experiment (MCRE)[22] generated using Cubit. The 2D circular section of the piping system is meshed using the `pave` scheme in Cubit, which produces an unstructured mesh composed by quadrilateral elements. Additionally, boundary layers are defined near the pipe walls, which facilitate the creation of structured meshes and enable the specification of the first element's distance from the wall, crucial for accurately applying turbulence wall functions. The resulting 2D mesh is then swept through the closed reactor loop. Following this, the external part of the reactor core is meshed using a `triprimitive` mesh scheme, which generates a triangular-prismatic mesh for surfaces with three logical sides. A subsequent 360-degree sweep completes the formation of the hexahedral mesh for the reactor core.

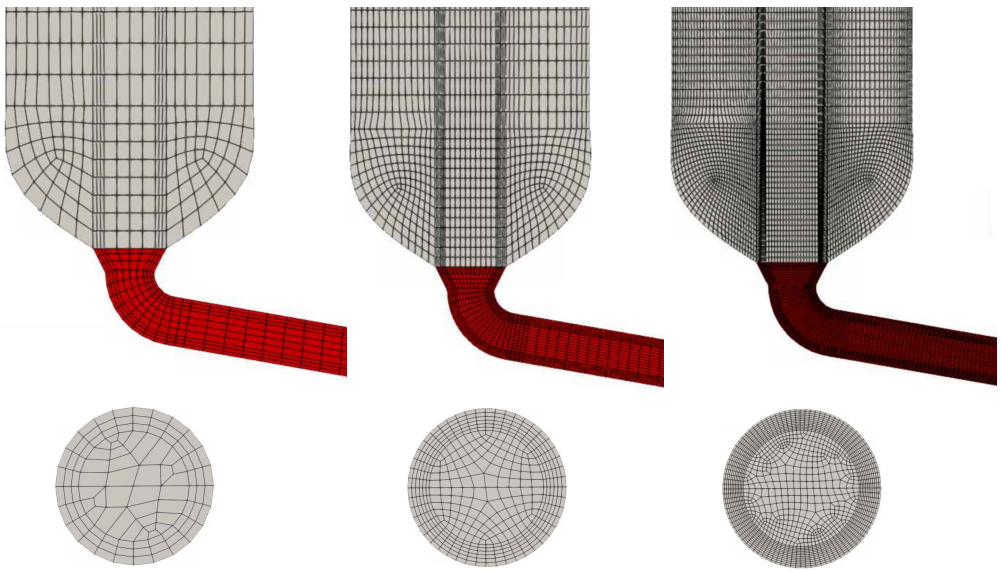

Figure 12. Successively refined Cubit-generated meshes for a molten salt reactor imported into MOOSE [2].

Figure 13 presents a 3D mesh of a valve of the Advanced Test Reactor (ATR). The mesh is cut at the middle plane of the valve which is in a fully closed position. The valve has six holes. The mesh was generated using directed mesh schemes with a 2D surface mesh in StarCCM+.

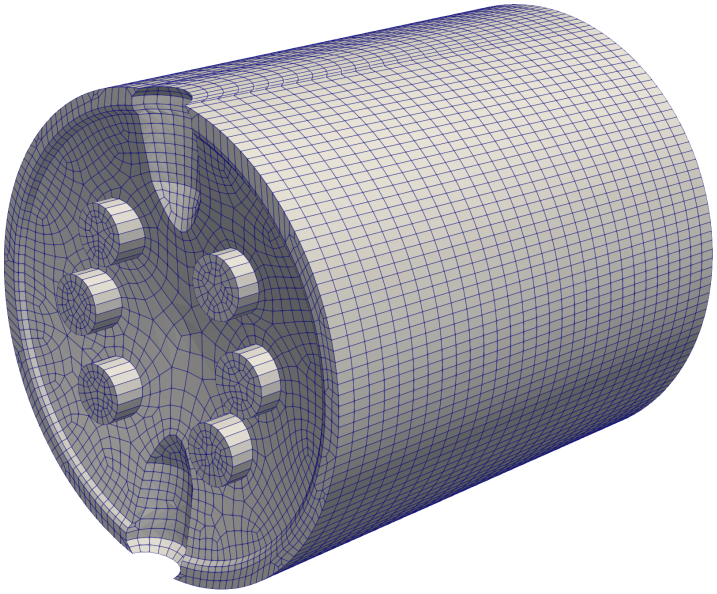

Figure 13. Mesh for the simulation of flow through the valve of the Advanced Test Reactor [6].

Figure 14 presents a 3D mesh of the Molten Salt Reactor Experiment (MSRE) prepared using the directed mesh capabilities in Cubit. It has been used for two-phase fluid dynamics simulations in [2].

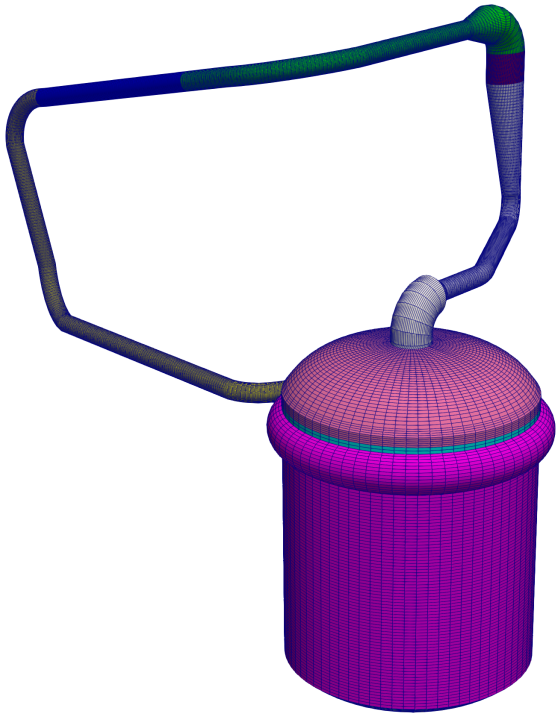

Figure 14. 3D mesh of the Molten Salt Reactor Experiment (MSRE) [2].

5.3. Future work

The current workflow for fluid dynamics simulations in MOOSE involves creating the geometry using CAD software. This is unlikely to change in the future. The path forward for the general needs of thermal hydraulics modeling in MOOSE must support CAD geometries. Additional prototyping of simple systems needs may rely on direct definition of the geometry in MOOSE.

5.3.1. Leveraging new capabilities from Netgen

While the meshing capabilities continue to grow in MOOSE, they are unlikely to be able to mesh general CAD geometries to the standard required by CFD simulations in the near term. MOOSE is able to leverage Netgen [42] for meshing arbitrary 3D regions using tetrahedral elements. As such, coarse tetrahedral meshes of general CAD are likely within reach. With existing element type conversion capabilities, and additional mesh smoothing capabilities, this could be a new pathway to generating meshes for general geometries.

5.3.2. Polyhedral element support

LibMesh [18] supports hexahedral, pyramidal, prismatic, and tetrahedral 3D elements. In CFD, polyhedras are also commonly used because of their capability for generating good quality meshes for arbitrary geometries. This meshing can be performed automatically, while meshing with hexahedral elements typically involves the user defining which surface to start from for every volume, or at least which algorithm to use for every region.

The quality of the simulations with polyhedral elements should be similar to using hexahedral meshes, but the simplification in the meshing workflow would be considerable. Adding support for polyhedral elements in libMesh would mean creating a new element class, as well as implementing many helper routines, quadrature rules and shape functions for these new elements. Once polyhedral elements are supported in libMesh, at least sufficiently for finite volume calculations, the next step should be to create readers for polyhedral mesh formats. This will enable reading polyhedral meshes created by external meshing software, notably from StarCCM+. While the Exodus format, commonly used in MOOSE, has been extended to support polyhedral elements, it is not commonly used by commercial tools. Several tools such as meshIO and ParaView offer format conversion options, so supporting a single additional format would likely be sufficient.

6. GEOMETRY AND PHYSICS ABSTRACTION FOR COMPONENTS

6.1. Abstract Component design

The `Components` system in MOOSE's Thermal Hydraulics module (THM) [12] allows users to conveniently build simulations composed of multiple pieces. For example, large reactor systems are composed of a network of pipes and reactor components and feature a variety of physical domains to model accurately. Each Component in THM may do one or more of the following, for example:

- Create a mesh (1D, 2D, or 3D).
- Add equations, source terms, and/or boundary conditions.
- Couple Components using interface terms.

Originally, the `Components` system in THM was designed to support only its own flow model, but it has since been sufficiently abstracted to allow applications to use the system with various models, whether they be flow models or some other physics. The abstraction is achieved using a generic `Component` class hierarchy that allows application developers to plug their own mesh and physics into Components. Figure 15 shows the hierarchy of `Component` base classes, which are all agnostic of physics.

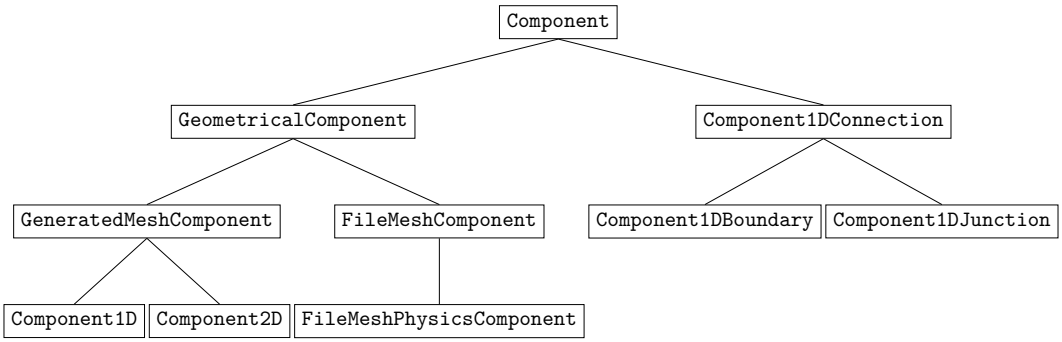

Figure 15. THM Component base class hierarchy.

These base classes are summarized as follows:

- `Component`: The base class of all Components.
- `GeometricalComponent`: The base class of all Components that have a 1D, 2D, or 3D mesh.
- `GeneratedMeshComponent`: The base class of all Components that generate their own mesh.
- `Component1D`: The base class of all Components that generate their own 1D mesh.
- `Component2D`: The base class of all Components that generate their own 2D mesh.
- `FileMeshComponent`: The base class of all Components that provide their mesh via a file.
- `FileMeshPhysicsComponent`: The base class of all Components that provide their mesh via a file and provide their physics via a Physics object (discussed later in this section).

- `Component1DConnection`: The base class of all Components that connect to end(s) of one or more 1D Components.
- `Component1DBoundary`: The base class of all Components that connect to one end of a 1D Component.
- `Component1DJunction`: The base class of all Components that connect to ends of two or more 1D Components.

Thus `Components` are organized by their geometrical relationship. There are options for both generating meshes and using existing meshes via files. With the exception of `FileMeshPhysicsComponent`, the classes listed here do not supply any physics. Physics can either be supplied external to the `Components` system, or one may inherit from one of these classes and add physics.

6.2. The Physics system

The `Physics` system, presented in [43] generally and [2] for Molten Salt Reactors physics, was designed to standardize the definition of equations, for every module and for both the regular MOOSE input syntax and the `Component` syntax. It is derived from the `Action` system. It has been widely deployed in the framework and tested in thermal-hydraulic models on the Virtual Test Bed [36].

`Physics` can be defined in a regular MOOSE simulation to replace the traditional `Variables`, `Kernels`, and `BoundaryConditions` objects. They can also be used in conjunction with a `FileMeshPhysicsComponent` to add an equation to be solved on a component geometrically defined by an external mesh file. The `Physics` available are shown in Table 8.

Table 8. Summary of Physics implemented in MOOSE and different MOOSE modules and applications.

Physics	Status	Notes
MOOSE		
Finite volume diffusion	available	—
Continuous Galerkin finite element (CGFE) diffusion	available	
Multi-species CGFE diffusion	available	
CGFE heat conduction	available	
Finite volume heat conduction	available	
Single-species trapping	pull request to TMAP8	
Single-species migration	pull request to TMAP8	
Navier Stokes with FVM		
Flow (mass and momentum equations)	available	Newton's method only
energy conservation	available	Newton's method only
scalar quantity conservation	available	Newton's method only
turbulence $k-\epsilon$	available	Newton's method only
Solid Mechanics		
Quasi-Static equations	available	syntax only
Dynamics equations	available	syntax only
Line Element quasi-static	available	syntax only
Cohesize Zone Model	available	syntax only

A notable absence from this list are flow models for thermal hydraulics. Using the `Physics` abstraction for flow models should be a priority for the upcoming fiscal year.

7. CONCLUSIONS

This work presents performance- and robustness-related improvements to the MOOSE-based thermal-hydraulics solvers along with meshing and component abstraction-related efforts.

The improved assembly algorithm resulted in a roughly $10\times$ speedup in computational time in the segregated solution routines in MOOSE's Navier-Stokes module. Even though the achieved speedup is significant compared to previous implementations, the algorithm is still slower than other commercial and open-source tools. The gap between these tools can be closed by several improvements, like assembling only one momentum matrix for the three momentum components. Nevertheless, this improvement clears the path towards using MOOSE for the engineering of Molten Salt Reactors (MSRs) which need higher-resolution velocity fields for accurately capturing corrosion effects and hot spots.

We presented two preconditioning techniques that can be applied to the monolithic solution algorithm with the finite volume method to reduce execution time and memory consumption. Both rely on the Schur complement-based field split preconditioner provided by PETSc. The first uses the pressure diagonal matrix as a preconditioner with a Successive Over-Relaxation algorithm. The second uses a HPDDM-based algorithm on the action of the full Schur complement. Both algorithms resulted in significant speedup in computation time and reduction in memory usage. These can enable scalable runs for larger problems using Newton's method.

The pseudo-time marching algorithm showed promising results as well. Two of the three implemented time step selection algorithms performed equally good or better than already existing time steppers in MOOSE in terms of computation time and robustness. At the same time, we note that the pseudo-time continuation seemed to exhibit sensitivity to the initial timestep size.

The improvements in meshing workflow involved the investigation on pathways to import meshes from StarCCM+, which allows the quick generation of high-quality meshing for complex domains. This accelerates the engineering workflows by saving time on model preparation. This is also an investment, considering that future work involves adding support for polyhedral meshes which can be automatically generated using these advanced tools.

The component abstraction in the Component system of the thermal-hydraulics module of MOOSE has been completed, effectively separating the geometry- and physics-related functionalities. This enables long term goals of having a common basis for multiple MOOSE-based thermal-hydraulics codes.

8. REFERENCES

[1] S. Schunert, A. Lindsay, P. German, J. Hansel, G. Giudicelli, M. Tano, L. Zou, and R. Hu, "Development plan for a common numerical layer for moose-based thermal-hydraulics codes (internal report)," tech. rep., Idaho National Laboratory, 2023.

[2] M. Tano, R. Freile, V. C. Leite, M. Li, J. Hansel, G. Giudicelli, and L. Charlot, "Advancing thermal-hydraulic modeling capabilities for molten salt reactors in pronghorn," Tech. Rep. INL/RPT-24-80305, Idaho National Laboratory, 2024.

[3] M. E. Tano Retamales, V. Kyriakopoulos, and S. Schunert, "Modeling of prismatic high temperature reactors in pronghorn," tech. rep., Idaho National Laboratory (INL), Idaho Falls, ID (United States), 2023.

[4] P. Balestra, S. Schunert, R. W. Carlsen, A. J. Novak, M. D. DeHart, and R. C. Martineau, "Pbmr-400 benchmark solution of exercise 1 and 2 using the moose based applications: Mammoth, pronghorn," in *EPJ Web of Conferences*, vol. 247, p. 06020, EDP Sciences, 2021.

[5] P. Wu, Y. Ma, C. Gao, W. Liu, J. Shan, Y. Huang, J. Wang, D. Zhang, and X. Ran, "Evaluating turbulence modeling for thermal-hydraulics analysis of molten salt reactors," *Nuclear Engineering and Design*, vol. 368, p. 110767, 2020.

[6] D. Yankura and M. Anderson, "Butterfly valve performance factors using the multiphysics object oriented simulation environment," *Annals of Nuclear Energy*, 2025.

[7] G. Giudicelli, A. Lindsay, L. Harbour, C. Icenhour, M. Li, J. E. Hansel, P. German, P. Behne, O. Marin, R. H. Stogner, J. M. Miller, D. Schwen, Y. Wang, L. Munday, S. Schunert, B. W. Spencer, D. Yushu, A. Recuero, Z. M. Prince, M. Nezdyur, T. Hu, Y. Miao, Y. S. Jung, C. Matthews, A. Novak, B. Langley, T. Truster, N. Nobre, B. Alger, D. Andrš, F. Kong, R. Carlsen, A. E. Slaughter, J. W. Peterson, D. Gaston, and C. Permann, "3.0 - MOOSE: Enabling massively parallel multiphysics simulations," *SoftwareX*, vol. 26, p. 101690, 2024.

[8] C. M. Rhie and W.-L. Chow, "Numerical study of the turbulent flow past an airfoil with trailing edge separation," *AIAA journal*, vol. 21, no. 11, pp. 1525–1532, 1983.

[9] C. T. Kelley and D. E. Keyes, "Convergence analysis of pseudo-transient continuation," *SIAM Journal on Numerical Analysis*, vol. 35, no. 2, pp. 508–523, 1998.

[10] Siemens Digital Industries Software, "Simcenter STAR-CCM+, version 2402," Siemens 2024.

[11] J. E. Matsson, *An introduction to ansys fluent 2023*. Sdc Publications, 2023.

[12] J. Hansel, D. Andrs, L. Charlot, and G. Giudicelli, "The MOOSE thermal hydraulics module," *Journal of Open Source Software*, vol. 9, no. 94, p. 6146, 2024.

[13] P. German, A. D. Lindsay, M. E. Tano Retamales, R. O. Freile, S. Schunert, and G. L. Giudicelli, "Development of Segregated Thermal-Hydraulics Solvers in MOOSE," tech. rep., Idaho National Laboratory (INL), Idaho Falls, ID (United States), 2024.

[14] 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.

[15] S. V. Patankar and D. B. Spalding, "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows," in *Numerical prediction of flow, heat transfer, turbulence and combustion*, pp. 54–73, Elsevier, 1983.

[16] H. Jasak, *Error analysis and estimation for the finite volume method with applications to fluid flows*. PhD thesis, Imperial College London (University of London), 1996.

[17] F. Juretic, *Error analysis in finite volume CFD*. PhD thesis, Imperial College London (University of London), 2005.

[18] 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. https://doi.org/10.1007/s00366-006-0049-3.

[19] A. Lindsay, R. Stogner, D. Gaston, D. Schwen, C. Matthews, W. Jiang, L. K. Aagesen, R. Carlsen, F. Kong, A. Slaughter, *et al.*, "Automatic differentiation in MetaPhysicL and its applications in MOOSE," *Nuclear Technology*, vol. 207, no. 7, pp. 905–922, 2021.

[20] T. D. Blacker, S. J. Owen, M. L. Staten, W. R. Quadros, B. Hanks, B. W. Clark, R. J. Meyers, C. Ernst, K. Merkley, R. Morris, *et al.*, "CUBIT geometry and mesh generation toolkit 15.1 user documentation," tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2016.

[21] J. P. Van Doormaal and G. D. Raithby, "Enhancements of the simple method for predicting incompressible fluid flows," *Numerical heat transfer*, vol. 7, no. 2, pp. 147–163, 1984.

[22] D. Walter, T. Cisnero, S. Goodrich, and Z. Mausolff, "Mcre design description in support of external model development," *TerraPower (Dec. 27, 2022)*, 2023.

[23] R. I. Issa, "Solution of the implicitly discretised fluid flow equations by operator-splitting," *Journal of computational physics*, vol. 62, no. 1, pp. 40–65, 1986.

[24] F. Nataf and P.-H. Tournier, "A geneo domain decomposition method for saddle point problems," *Comptes Rendus. Mécanique*, vol. 351, no. S1, pp. 667–684, 2023.

[25] Y. He and M. Olshanskii, "A preconditioner for the grad-div stabilized equal-order finite elements discretizations of the oseen problem," *arXiv preprint arXiv:2407.07498*, 2024.

[26] H. Elman, V. E. Howle, J. Shadid, D. Silvester, and R. Tuminaro, "Least squares preconditioners for stabilized discretizations of the navier–stokes equations," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 290–311, 2008.

[27] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, *et al.*, "PETSc users manual," 2019.

[28] M. Benzi and A. J. Wathen, "Some preconditioning techniques for saddle point problems," *Model order reduction: theory, research aspects and applications*, pp. 195–211, 2008.

[29] P. Ghysels, X. S. Li, C. Gorman, and F.-H. Rouet, "Strumpack: Scalable preconditioning using low-rank approximations and random sampling," *SC'16*, 2016.

[30] U. M. Yang *et al.*, "Boomeramg: A parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155–177, 2002.

[31] P. Jolivet, F. Hecht, F. Nataf, and C. Prud'Homme, "Scalable domain decomposition preconditioners for heterogeneous elliptic problems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, 2013.

[32] X. He, C. Vuik, and C. Klaij, "Block-preconditioners for the incompressible Navier–Stokes equations discretized by a finite volume method," *Journal of Numerical Mathematics*, vol. 25, no. 2, pp. 89–105, 2017.

[33] H. M. Bücker, B. Pollul, and A. Rasch, "On CFL evolution strategies for implicit upwind methods in linearized Euler equations," *International journal for numerical methods in fluids*, vol. 59, no. 1, pp. 1–18, 2009.

[34] M. Ceze and K. Fidkowski, "Pseudo-transient continuation, solution update methods, and CFL strategies for DG discretizations of the RANS-SA equations," in *21st AIAA computational fluid dynamics conference*, p. 2686, 2013.

[35] A. Abou-Jaoude, S. Harper, G. Giudicelli, P. Balestra, S. Schunert, N.Martin, A. Lindsay, and M. Tano, "A Workflow Leveraging MOOSE Transient Multiphysics Simulations to Evaluate the Impact of Thermophysical Property Uncertainties on Molten-Salt Reactors," *Annals of Nuclear Energy*, vol. 163, p. 108546, 2021.

[36] G. L. Giudicelli, A. Abou-Jaoude, A. J. Novak, A. Abdelhameed, P. Balestra, L. Charlot, J. Fang, B. Feng, T. Folk, R. Freile, T. Freyman, D. Gaston, L. Harbour, T. Hua, W. Jiang, N. Martin, Y. Miao, J. Miller, I. Naupa, D. O'Grady, D. Reger, E. Shemon, N. Stauff, M. Tano, S. Terlizzi, S. Walker, and C. Permann, "The virtual test bed (VTB) repository: A library of reference reactor models using neams tools," *Nuclear Science and Engineering*, vol. 0, no. 0, pp. 1–17, 2023.

[37] E. Shemon, Y. Miao, S. Kumar, K. Mo, Y. S. Jung, A. Oaks, S. Richards, G. Giudicelli, L. Harbour, and R. Stogner, "MOOSE Reactor Module: An Open-Source Capability for Meshing Nuclear Reactor Geometries," *Nuclear Science and Engineering*, vol. 0, no. 0, pp. 1–25, 2023.

[38] M. E. Tano Retamales, A. Karahan, A. Novak, and S. Schunert, "Development of a subchannel capability for liquid-metal fast reactors in pronghorn," tech. rep., Idaho National Laboratory (INL), Idaho Falls, ID (United States), 2022.

[39] INL, "MOOSE File Mesh Generator." https://mooseframework.inl.gov/source/meshgenerators/FileMeshGenerator.html. [Online; accessed 09-16-2024].

[40] A. Ribes and C. Caremoli, "Salome platform component model for numerical simulation," in *31st annual international computer software and applications conference (COMPSAC 2007)*, vol. 2, pp. 553–564, IEEE, 2007.

[41] C. Geuzaine and J.-F. Remacle, "Gmsh: a three-dimensional finite element mesh generator with built-in pre-and post-processing facilities," 2008.

[42] J. Schöberl, "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules," *Computing and Visualization in Science*, vol. 1, pp. 41–52, 1997.

[43] A. Lindsay, R. Stogner, G. Giudicelli, M. Li, L. Harbour, and C. Permann, "Increased accuracy of multiphysics simulations through flexible execution, transient algorithms, and modular physics," Tech. Rep. INL/RPT-24-78792, Idaho National Laboratory, 2024.