Full-core multiphysics analysis of Molten Salt Reactor Experiment using parallel code

Andrei Rykhlevskii

andreir2@illinois.edu

INTRODUCTION

The Molten Salt Reactor (MSR) is an advanced type of reactor which was developed at Oak Ridge National Laboratory (ORNL) in the 1950s and was operated in the 1960s. In the MSR, fluorides of fissile and/or fertile materials (i.e. UF_4, PuF_3 and/or ThF_4) are mixed with carrier salts to form a liquid fuel which is circulated in a loop-type primary circuit [1]. This innovation leads to immediate advantages over traditional, solid-fueled, reactors. These include near-atmospheric pressure in the primary loop, relatively high coolant temperature, outstanding neutron economy, a high level of inherent safety, reduced fuel preprocessing, and the ability to continuously remove fission products and add fissile and/or fertile elements [2].

LITERATURE REVIEW

MSR modeling efforts describe steady-state and transient behavior. Krepel et al. extended the in-house Light Water Reactor (LWR) diffusion code DYN3D to consider drift of delayed neutron precursors alongside the reactor temperature profile, re-casting the extended code as DYN3D-MSR [3]. That work compared DYN3D-MSR against experimental Molten Salt Reactor Experiment (MSRE) data and then used it to simulate local fuel channel blockages as well as local temperature perturbations. In a similar vein, Kophazi et. al. used iterative coupling between in-house three-dimensional neutronic and one-dimensional heat conduction models DALTON and THERM to analyze normal MSRE operation as well as channel-blocking-incident transients [4]. The Kophazi model added entrance effects of heat transfer coefficients as well as thermal coupling between fuel channels through moderator heat conduction. More recently, Cammi et. al. performed a 2D-axisymmetric single-channel analysis of the Molten Salt Breeder Reactor (MSBR) using the commercial finite element package COMSOL Multiphysics [5]. That work directly solved the fuel salt velocity field, used heterogeneous group constants in fuel and moderator regions, and employed a software package (COMSOL) intrinsically designed for coupled multi-physics simulation. Additionally, Aufiero et. al [6] have begun to approach transient simulations in the Molten Salt Fast Reactor (MSFR) by directly coupling Serpent 2 Monte Carlo neutronics with OpenFOAM [7] thermal-hydraulics.

APPROACH AND METHOD

Recently developed multiphysics code Moltres [8] could be employed for simulating MSRs. By implementing deterministic neutronics and thermal hydraulics in the context of the Multiphysics Object-Oriented Simulation Environment (MOOSE) finite element modeling framework, Moltres solves arbitrary-group neutron diffusion, temperature, and precursor governing equations in anywhere from one to three dimensions and can be deployed on an arbitrary number of processing units. Through its computing power, Moltres is devoted to previously unmatched fidelity in coupled neutronics and thermal hydraulics MSR simulation.

Moreover, Moltres depends on the MOOSE framework, [9] which is Lesser GNU Public License (LGPL) code that itself leans on LibMesh [10], a LGPL finite element library, and PetSc [11], a Berkeley Software Distribution (BSD)-licensed toolkit for solving nonlinear equations yielded by discretizing PDEs. MOOSE and LibMesh translate weak PDE forms defined by applications (e.g. Moltres) into residual and Jacobian functions. These functions are the inputs into PetSc Newton-Raphson solution routines. All codes use MPI for parallel communication and are easily deployed on massively-parallel cluster-computing platforms. MOOSE applications by default use monolithic and implicit methods ideal for closely-coupled and multi-scale physics, such as the model problem described in this work. However, Moltres can also use explicit time-stepping routines as well as segregated solution methods, making it extensible to myriad future modeling challenges.

In Moltres, neutrons are described with time-dependent multi-group diffusion theory:

$$\frac{1}{v_g}\frac{\partial \phi_g}{\partial t} - \nabla \cdot D_g \nabla \phi_g + \Sigma_g^r \phi_g = \chi_g^p \sum_{g'=1}^{G} (1-\beta)\nu \Sigma_{g'}^f \phi_{g'} +$$

$$\sum_{g \neq g'}^{G} \Sigma_{g' \to g}^s \phi_{g'} + \chi_g^d \sum_{i}^{I} \lambda_i C_i \qquad (1)$$

v_g = speed of neutrons in group g
ϕ_g = flux of neutrons in group g
t = time
D_g = diffusion coefficient for neutrons in group g
Σ_g^r = macro removal cross-section from group g
$\Sigma_{g' \to g}^s$ = macroscopic cross-section of scattering from g' to g
χ_g^p = prompt fission spectrum, neutrons in group g
G = number of discrete groups, g
ν = number of neutrons produced per fission
Σ_g^f = macro cross-section for fission in group g
χ_g^d = delayed fission spectrum, neutrons in group g
I = number of delayed neutron precursor groups
β = delayed neutron fraction
λ_i = average decay constant of delayed neutron precursors in i
C_i = concentration of delayed neutron precursors in group i

Delayed neutron precursors are described by equation:

$$\frac{\partial C_i}{\partial t} = \sum_{g'=1}^{G} \beta_i \nu \Sigma_{g'}^f \phi_{g'} - \lambda_i C_i - \frac{\partial}{\partial z} u C_i \qquad (2)$$

with the last term representing the effect of fuel advection. The governing equation for the temperature is given by:

$$\rho_f c_{p,f}\frac{\partial T_f}{\partial t} + \nabla\cdot\left(\rho_f c_{p,f}\boldsymbol{u}\cdot T_f - k_f\nabla T_f\right) = Q_f \qquad (3)$$

ρ_f = density of fuel salt
$c_{p,f}$ = specific heat capacity of fuel salt
T_f = temperature of fuel salt
$\boldsymbol{u}$ = velocity of fuel salt
k_f = thermal conductivity of fuel salt
Q_f = source term

in the fuel, where the source term Q_f is defined by:

$$Q_f = \sum_{g=1}^{G}\epsilon_{f,g}\Sigma_{f,g}\phi_g \qquad (4)$$

In the moderator, the governing equation for temperature is given by:

$$\rho_g c_{p,g}\frac{\partial T_g}{\partial t} + \nabla\cdot\left(-k_g\nabla T_g\right) = Q_g \qquad (5)$$

Group constants must be generated by the modeler with either Serpent [12] or SCALE [13]. Moltres interpolates group constant temperature dependence from prepared tables, which must be constructed separately for fuel and moderator regions.

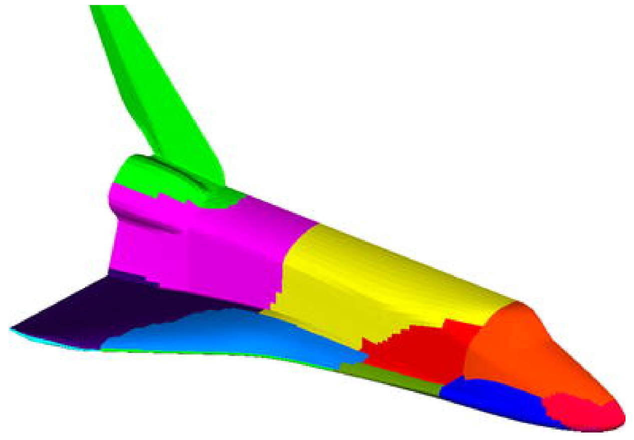

Fig. 1: Element-based domain decomposition of a surface mesh into 16 subdomains [14].

OBJECTIVES

In this study set of equations (1-5) in 2-D geometry (r-z) for well-studied MSRE was solved using MOOSE-based multiphysics code Moltres and results were compared with reference [15]. Moltres predicted temperature profiles considered with cosinusoidal gamma heating for the hottest channel and adjacent graphite. It should be noted that the ORNL MSRE design calculations, conducted in 1963-1964, were 32-group calculations using legacy computing tools (GAM-I, MODRIC, and EQUIPOSE, and THERMOS). Those calculations were conducted in two-dimensional R-Z geometry (a cylinder with angular symmetry), with 20 spatial regions.

Secondly, strong and weak scaling study was performed to find out appropriateness using Moltres for heavy computing full-core analysis and estimate "sweet" spot for large-scale calculations. Moreover, the results of scaling study clearly show the limitation of parallelization methods used in Moltres.

Finally, relative performance monolithic vs. segregated solution was considered. We usually use Moltres multiphysics code in the monolithic mode when it solves all equations in one element and after convergence switch to next element. To verify the relative efficiency of this approach finite-difference parallel solver for the simplified 2-D case was developed to solve one-group neutron diffusion equation for zero nuclear data and vacuum boundary conditions to estimate strong and weak scaling efficiency to compare with Moltres. For fair comparison (Moltres solves 5 equations while developed solver only 1) execution time for segregated solver was multiplied by 5.

PARALLELIZATION

MOOSE framework uses LibMesh finite element library to solve set of physics equations. Different finite element formulations may be applied including Galerkin, Petrov-Galerkin, and discontinuous Galerkin methods [14]. Parallelism is achieved using domain decomposition through mesh partitioning, in which each processor contains the global mesh but in general computes only on a particular subset. Parallel implicit linear systems are supported via an interface with the PETSc library.

A standard non-overlapping domain decomposition approach is used in LibMesh to achieve data distribution on parallel computers as shown in Fig. 1. The elements in each subdomain are assigned to an individual processor. The two primary metrics in judging the quality of a partition are the subdomain mesh size and the number of "edge cuts" in the resulting partition. For a mesh composed of a single type of element, each subdomain should contain an equal number of elements so that the resulting domain decomposition is load balanced across all available processors. Usually, for MSR simulation adaptive mesh refinement and coarsening (AMR/C) scheme provided by LibMesh is employed. AMR/C allows do not explicitly specify domain decomposition, and provides optimal parallel performance with minimal user's efforts.

Developed C++/MPI code has using finite-difference iterative method with a fixed number of iterations (>4000) to solve one-group neutron diffusion equation in a 2-D mesh. For simplicity 2-D Poisson's equation which is similar to one-group neutron diffusion equation was considered. At almost every grid point (i,j) the finite difference approximation to Poisson's equation is:

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2}) = -f(x_i, y_j) \qquad (6)$$

Five-point stencil of the 2-D finite difference approximation shown in Fig. 2

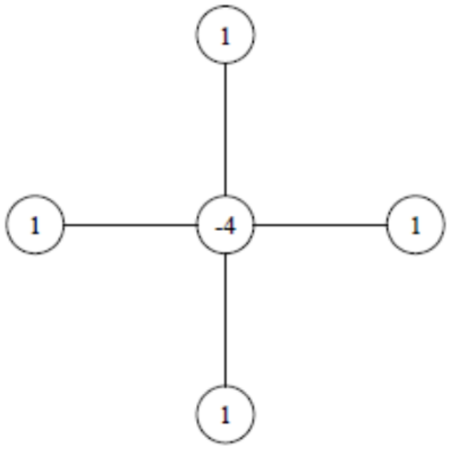

Fig. 2: 5-point stencil for 2-D finite difference approximation.

Natural row-wise ordering is completely sequential, hence, reordering methods required. For Jacobi interative method best reordering strategy is red-black ordering when color the alternate grid points in each dimension red or black (Fig. 3).

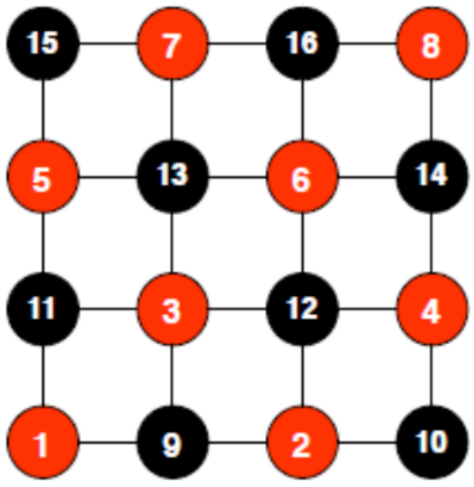

Fig. 3: Red-Black Ordering.

To implement selected reordering strategy first iterates on red grid point only:

$$u_{i,j}^{(k+1)} = (1-w)u_{i,j}^{(k)} + w/4(h^2 f_{i,j} + u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)}) \tag{7}$$

Then iterates on black points only:

$$u_{i,j}^{(k+1)} = (1-w)u_{i,j}^{(k)} + w/4(h^2 f_{i,j} + u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)}) \tag{8}$$

After that algorithm sends/receives values of the black-points at the boarder of the subdomain to neighboring processes and compute residual to check convergence (for scaling study number of iterations was fixed).

Preconditioned Conjugate Gradient (CG) method was used to accelerate convergence rate. Parallelization implemented using 2-D sub-matrices decomposition, when the 2D domain is divided into multiple sub-domains depending on the available number of computer nodes Fig. 4 For interior points, neutron diffusion equation will be solved implicitly by Jacobi iterative scheme in combining with the boundary conditions. At the interface points of interior subdomains, the equation will be solved by explicit iterative schemes. The proposed approach fulfills the suitability for the implementation on Linux PC cluster through the minimization of inter-process communication by restricting the exchange of data to the interface between the sub-domains. To examine the efficiency and accuracy of the iterative algorithm, several numerical experiments using different number of nodes of the Linux PC cluster was conducted. The performance metrics should clearly show the benefit of using multicore system in terms of execution time reduction and speedup with respect to the sequential running in a single core.

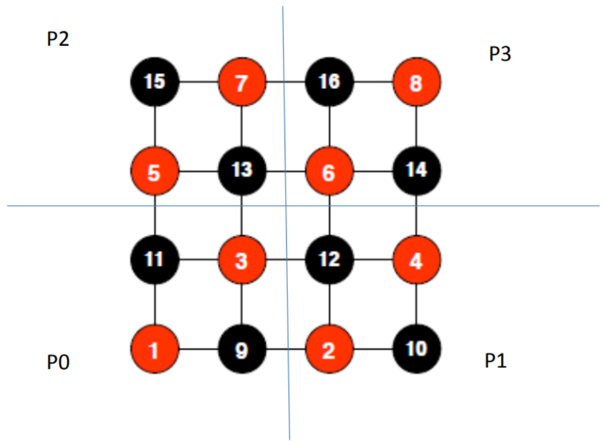

Fig. 4: Red-Black Ordering parallelization.

RESULTS

Comparison with MSRE

Fig. 5 shows a comparison between Moltres predicted temperature and MSRE design calculations [15] in the hottest channel and adjacent graphite.

The profile shapes are in decent qualitative agreement with both Moltres and MSRE calculations showing a peak in graphite temperature before the reactor outlet. Fuel temperature increases monotonically in both Moltres and MSRE models. In the MSRE design, the moderator temperature at the reactor inlet is about 11 K larger than the fuel temperature, whereas the temperatures are about the same in the Moltres model. This difference is likely because the MSRE design model neglected axial heat conduction [15, p. 99].

Fig. 6 compares the fast and thermal neutron fluxes at the reactor mid-plane ($z = H/2$) for Moltres and MSRE design models. Local thermal flux growth and fast flux decay in moderator regions and visa versa in fuel regions are apparent in the Moltres calculation. The Moltres flux magnitudes are in good agreement with the magnitudes from the MSRE design calculations [15, p. 92]. The peak fast to thermal flux ratio is approximately 3.5 in the MSRE design calculation as opposed to a ratio of 3 for the Moltres calculation. Control rod thimbles and an extra volume of surrounding fuel not included in the

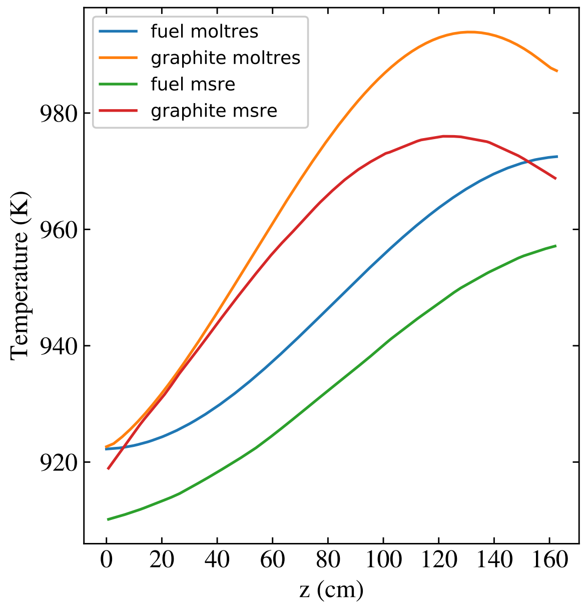

Fig. 5: Moltres and MSRE design [15, p. 99] predicted axial temperature profiles in hottest channel and adjacent graphite

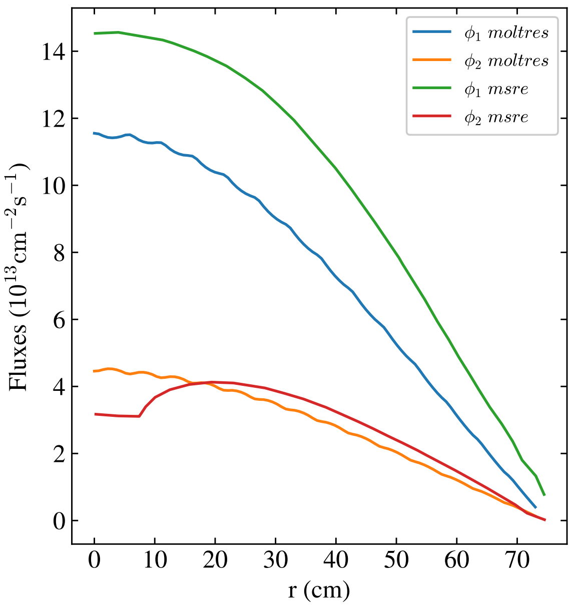

Fig. 6: The thermal and fast flux profiles at the core mid-plane ($z = H/2$) for the Moltres 2-D cylindrical axisymmetric model and the MSRE design model [15, p. 92] ($r = 0$ is radial center of core).

Moltres calculations cause the depression in the thermal flux in the MSRE profile.

Fig. 7 compares the axial flux profiles calculated by Moltres and the ORNL MSRE design model. The radii for the plots are chosen to correspond to the peak of the thermal flux in both cases; for the ORNL calculations this is 21.336 cm (8.4 inches) from the core center-line because of the effect of the control rod thimbles and extra fuel along the center-line. Once again, the plots are in decent agreement. The ORNL calculations include the lower and upper plena which are not included in this report's Moltres model. Consequently, the MSRE lines extend to lower and higher z-values than the Moltres lines. Additionally, absorption in the plena cause deviation of the thermal flux from a sinusoidal shape in the MSRE design case. The peak power density from the MSRE calculation is 31 kW/L; the corresponding value for Moltres is 29 kW/L.

Moltres scaling study

Parallelization in Moltres is implemented via LibMesh, which includes a set of utilities for massively parallel finite element based computations, including mesh input/output, a finite element library, and an interface for connections with solver packages. Employing LibMesh provides Moltres with significant flexibility including the ability to swap out solver libraries such as PetSc, which includes an expanding suite of parallel linear and nonlinear solvers. Problem domain decomposition relies on LibMesh mesh adaptation capabilities for running on a specific number of processors and can either be performed manually before the start of the simulation or automatically at the parameters of computation.

We conducted strong and weak scaling studies to characterize parallel performance in Moltres. In case of strong scaling, the problem size remains fixed but the number of pro-

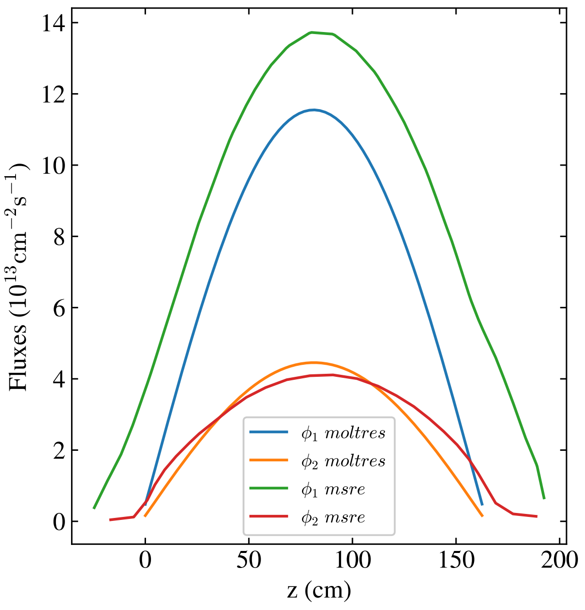

Fig. 7: Moltres axial flux profiles along the core center line and MSRE design axial flux profiles 21.336 cm (8.4 inches) from the core center line [15, p. 91].

cessors is increased. Strong scaling studies seek to identify an optimal ratio between the number of processors and elements for the most rapid and power-efficient computation for a given problem. We measured Moltres strong scaling with a simple 2D axisymmetric case for various problem sizes separately for intra-node (2,820; 5,640; 11,280 and 28,200 elements) and extra-node (86,655; 173,310; 317,735; 664,355 elements) setup on Blue Waters' XK7 nodes (two AMD 6276 Interlagos CPU per node, 16 floating-point bulldozer core units per node or 32 "integer" cores per node, nominal clock speed is 2.45 GHz).

Fig. 8 shows the simulation speed in seconds per element vs. the number of cores on 1 node (maximum 32 cores). Up to 8 cores, larger problems required considerably more time per element because of cache overhead. However, beyond 8 cores, scaling demonstrates asymptotic dependence on the number of processors due to increasing communication costs. The best parallel efficiency for the intra-node study is approximately 89% and has been achieved for the largest problem (28,200 elements).

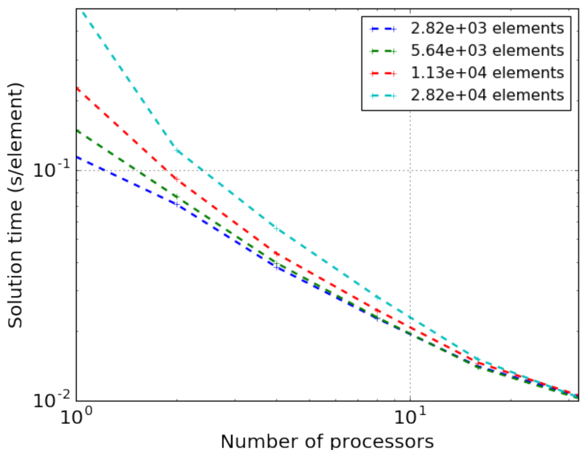

Fig. 8: Moltres intra-node strong scaling efficiency for various problem sizes, for $n_{cores} \in [1, 32]$.

Fig. 9 shows Moltres strong scaling up to 768 processors. This takes into account communication costs between nodes. Similar to the intra-node study, when fewer than 128 cores were used, cache overhead causes performance slow down for larger problems. However, beyond 256 cores, the simulation time per element remains almost constant for small cases (86,655 and 173,310 elements) and slighly decreases for the two larger problems. For extra-node scaling, parallel efficiency also grows with the problem size and reaches an optimal value of 73% for 664,355 elements.

For the weak scaling study, the part of the problem (workload) assigned to each processor stays constant and additional elements are used to solve a bigger problem which would not fit in memory on a single node. Thus, the weak scaling measurement is justification for memory-bound application such as multiphysics code. Linear weak scaling is achieved when the execution time stays constant while the workload increasing in direct proportion to the number of cores. We

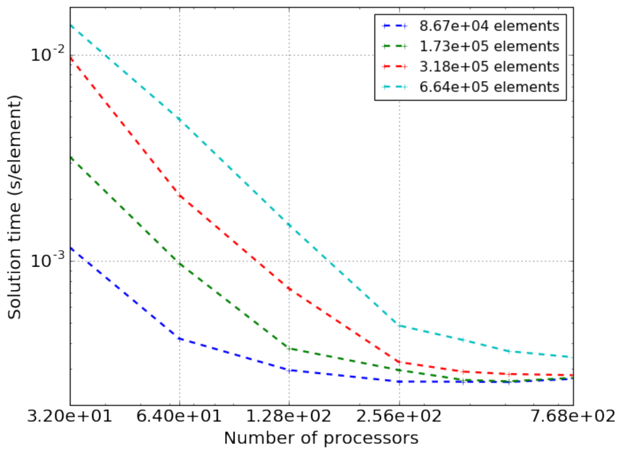

Fig. 9: Moltres extra-node strong scaling efficiency for various problem size, for $n_{nodes} \in [1, 24]$.

performed Moltres weak-scaling tests on Blue Waters, keeping the workload constant at 581, 985, 1970 and 3940 elements per core. Fig. 10 shows Moltres weak scaling performance measured for $n_{cores} \in [1, 32]$ within one Blue Waters node and Fig. 11 demonstrates performance for $n_{cores} \in [32, 128]$. As expected, the largest drop in performance occurs when the number of cores increases from one to ≈ 8, which corresponds to switching from no communication to a 2-D domain decomposition. The further reduction in performance of only about 50% over a range of 32 cores is likely caused by increased communication latency appearing from collective Message Passing Interface (MPI) calls. In the extra-node case, the performance drops by a factor of three, which is most likely due to poor node selection by the Blue Waters job scheduler and significantly increased latency and bandwidth costs.

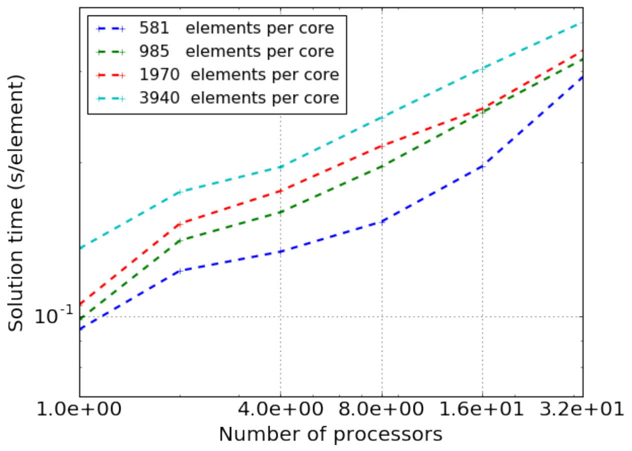

Fig. 10: Weak scaling performance of Moltres on Blue Waters, in seconds per element vs. number of processors, for a constant number of elements per processor, and $n_{cores} \in [1, 32]$.

Moltres scalability study results clearly indicate that par-

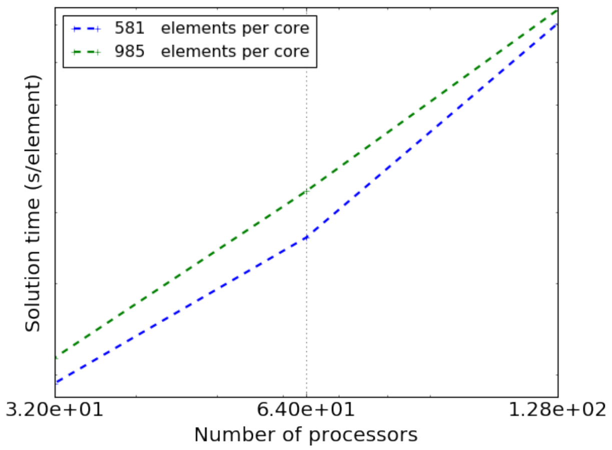

Fig. 11: Weak scaling performance of Moltres on Blue Waters, in seconds per element vs. number of processors, for a constant number of elements per processor, for and $n_{cores} \in [32, 128]$.

allelization using LibMesh's automatic domain decomposition is good, but not perfectly efficient. This scaling performance is satisfactory for MSR simulations approached thus far and improved parallel performance would require further optimization within LibMesh. Moreover, Moltres is memory-bound and therefore very sensitive to host memory and memory bandwidth. Consequently, if improved performance is needed, one could consider a transition from CPUs computing to GPU-accelerated computing because GPUs operate on the fly with global memory, avoiding CPU cache storage issues. Another way to improve parallel performance is to force the solver to use "older" information from previous iterations. However, this has been shown to slow convergence in terms of iterations and increased workload [11].

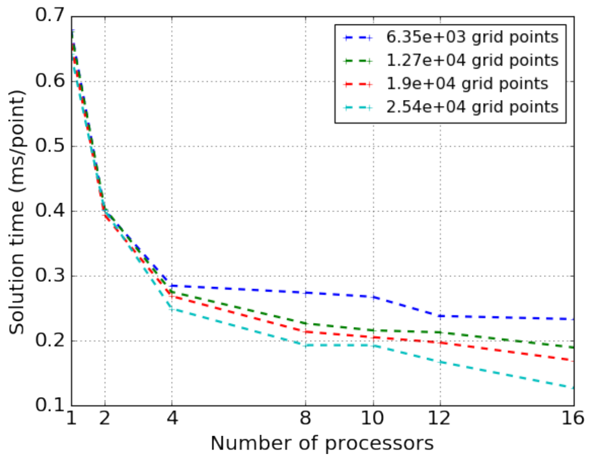

Fig. 12: Strong scaling execution time per grid point for 2-D Poisson's equation solver.

Parralel preconditioned Conjugate Gradient solver scaling study

As discussed earlier, to evaluate parallelization efficiency of Moltres multiphysics code simple solver was developed based on preconditioned Conjugate Gradient (CG) iterative method. Classic Jacobi preconditioner was used to accelerate convergence.

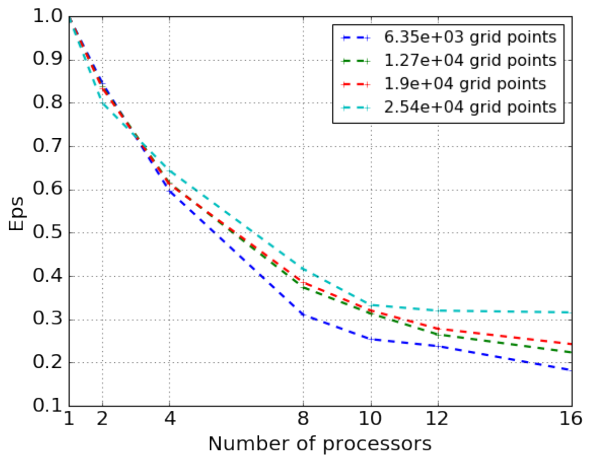

Fig. 13: Strong scaling parallel efficiency for 2-D Poisson's equation solver.

We conducted strong and weak scaling studies to characterize parallel performance for the iterative solver. Fig. 12 shows the simulation speed in milliseconds per grid point vs. the number of cores on 1 node (maximum 16 cores). Up to 4 cores, it shows good speedup, while beyond 4 cores, scaling demonstrates asymptotic dependence on the number of processors due to increasing communication costs. From fig. 13 could be observed that the parallel efficiency for the strong scaling study is decreasing dramatically for the case from 1 to 0.2 for the smallest input (6348 points) and in the range from 1 to 0.33 for the largest input (25'392 points). Such a horrible parallel efficiency most likely happens due to a fixed number of iteration (>10000) instead of using Jacobi or Gauss-Seidel methods and will be fixed in a future version of the solver.

Fig. 14,13 demonstrates results of weak scaling study when input size was fixed (1587 points per processor). Linear weak scaling is achieved when the execution time stays constant while the workload increasing in direct proportion to the number of cores. The largest drop in performance occurs when the number of cores increases from one to ≈ 8, which corresponds to switching from no communication to a 2-D domain decomposition. The further reduction in performance is only about 10% over a range of 16 cores is likely caused by increased communication latency appearing from collective MPI calls. Similarly to strong scaling case, weak scaling parallel efficiency drops significantly from 1.0 to 0.3 most likely because the very small input was selected. For the few orders of magnitude larger workload efficiency of weak scaling expected to be much better. Moreover, Jacobi or Gauss-Seidel convergence checker will be implemented as discussed earlier for strong scaling study.

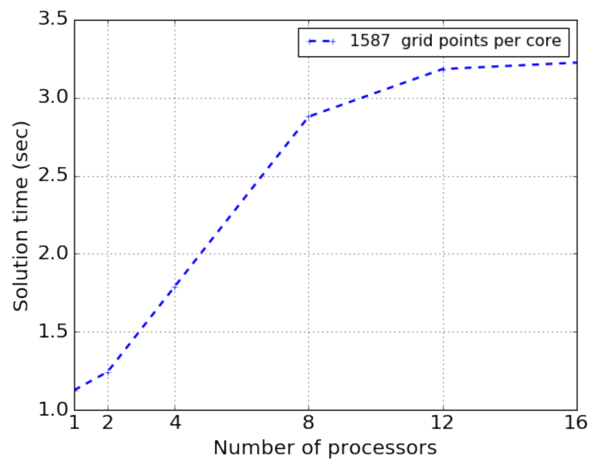

Fig. 14: Weak scaling execution time for 2-D Poisson's equation solver.

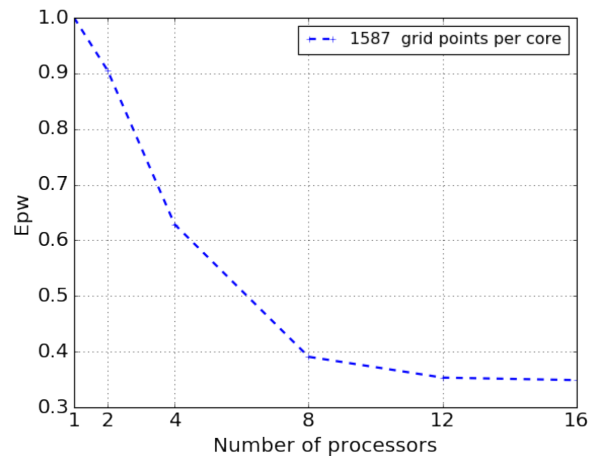

Fig. 15: Weak scaling parallel efficiency for 2-D Poisson's equation solver.

FUTURE WORK

- Detail theoretical computational cost analysis will be performed to find sources of overhead.
- Work underway to coupled multigroup diffusion and discontinuous Galerkin precursor transport to Navier-Stokes modules.
- Effects of salt/graphite temperature change cross terms yet to be quantified.
- Parallelization optimization tweaking parameters of preconditioners and solvers in PETSc (i.e. ILU parameters).
- Precoursors decay heat implementation (expected 8% of total power).
- Natural convection implementation using the Boussinesq approximation.
- Transition from CPUs computing to GPU-accelerated computing to avoid CPU cache storage issues.

REFERENCES

1. P. N. HAUBENREICH and J. R. ENGEL, "Experience with the Molten-Salt Reactor Experiment," *Nuclear Technology*, **8**, *2*, 118–136 (Feb. 1970).
2. D. LEBLANC, "Molten salt reactors: A new beginning for an old idea," *Nuclear Engineering and Design*, **240**, *6*, 1644–1656 (Jun. 2010).
3. J. KÅŹEPEL, U. ROHDE, U. GRUNDMANN, and F.-P. WEISS, "DYN3D-MSR spatial dynamics code for molten salt reactors," *Annals of Nuclear Energy*, **34**, *6*, 449–462 (Jun. 2007).
4. J. KOPHAZI, D. LATHOUWERS, and J. KLOOSTERMAN, "Development of a Three-Dimensional Time-Dependent Calculation Scheme for Molten Salt Reactors and Validation of the Measurement Data of the Molten Salt Reactor Experiment," *Nuclear Science and Engineering*, **163**, *2*, 118–131 (2009).
5. A. CAMMI, V. DI MARCELLO, L. LUZZI, V. MEMOLI, and M. E. RICOTTI, "A multi-physics modelling approach to the dynamics of Molten Salt Reactors," *Annals of Nuclear Energy*, **38**, *6*, 1356–1372 (Jun. 2011).
6. M. AUFIERO, A. CAMMI, O. GEOFFROY, M. LOSA, L. LUZZI, M. E. RICOTTI, and H. ROUCH, "Development of an OpenFOAM model for the Molten Salt Fast Reactor transient analysis," *Chemical Engineering Science*, **111**, 390–401 (May 2014).
7. H. G. WELLER, G. TABOR, H. JASAK, and C. FUREBY, "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in physics*, **12**, *6*, 620–631 (1998).
8. A. LINDSAY, "Moltres, a code for simulating Molten Salt Reactors," (2017), https://github.com/arfc/moltres.
9. D. R. GASTON, C. J. PERMANN, J. W. PETERSON, A. E. SLAUGHTER, D. ANDRÅĄ, Y. WANG, M. P. SHORT, D. M. PEREZ, M. R. TONKS, J. ORTENSI, L. ZOU, and R. C. MARTINEAU, "Physics-based multiscale coupling for full core nuclear reactor simulation," *Annals of Nuclear Energy*, **84**, 45–54 (Oct. 2015).
10. 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*, **22**, *3-4*, 237–254 (Dec. 2006).
11. SATISH BALAY, SHRIRANG ABHYANKAR, MARK ADAMS, JED BROWN, PETER BRUNE, KRIS BUSCHELMAN, LISANDRO DALCIN, VICTOR EIJKHOUT, WILLIAM GROP, DINESH KAUSHIK, MATTHEW KNEPLEY, LOIS CURFMAN MCINNES, KARL RUPP, BARRY SMITH, STEFANO ZAMPINI, and HONG ZHANG, "PETSc Users Manual," Tech. Rep. ANL-95/11 - Revision 3.6, Argonne National Laboratory (2015).
12. J. LEPPÄNEN, M. PUSA, T. VIITANEN, V. VALTAVIRTA, and T. KALTIAISENAHO, "The Serpent Monte Carlo code: Status, development and applications in 2013," *Annals of Nuclear Energy*, **82**, 142–150 (Aug. 2015).
13. M. D. DEHART and S. M. BOWMAN, "Reactor Physics Methods and Analysis Capabilities in SCALE," *Nuclear*

Technology, **174**, *2*, 196–213 (May 2011).

14. 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*, **22**, *3*, 237–254 (Dec 2006).
15. R. B. BRIGGS, "Molten-Salt Reactor Program semiannual progress report for period ending July 31, 1964," Technical Report Archive and Image Library ORNL-3708, Oak Ridge National Laboratory, Oak Ridge, TN, United States (1964).