EXASCALE COMPUTING PROJECT

ECP Milestone Report

High-order algorithmic developments and optimizations for large-scale GPU-accelerated simulations

WBS 2.2.6.06, Milestone CEED-MS36

Tzanio Kolev
Paul Fischer
Anthony P. Austin
Andrew T. Barker
Natalie Beams
Jed Brown
Jean-Sylvain Camier
Noel Chalmers
Veselin Dobrev
Yohann Dudouit
Leila Ghaffari
Stefan Kerkemeier
Yu-Hsiang Lan
Elia Merzari
Misun Min
Will Pazner
Thilina Rathnayake
Mark S. Shephard
Morteza H. Siboni
Cameron W. Smith
Jeremy L. Thompson
Stanimire Tomov
Tim Warburton

March 31, 2021

ECP Milestone Report
High-order algorithmic developments and optimizations for large-scale GPU-accelerated simulations
WBS 2.2.6.06, Milestone CEED-MS36

Office of Advanced Scientific Computing Research
Office of Science
US Department of Energy

Office of Advanced Simulation and Computing
National Nuclear Security Administration
US Department of Energy

March 31, 2021

ECP Milestone Report
High-order algorithmic developments and optimizations for large-scale GPU-accelerated simulations
WBS 2.2.6.06, Milestone CEED-MS36

Approvals

Submitted by:

Tzanio Kolev, LLNL CEED PI	Date

Approval:

Andrew R. Siegel, Argonne National Laboratory Director, Applications Development Exascale Computing Project	Date

Revision Log

Version	Creation Date	Description	Approval Date
1.0	March 31, 2021	Original	

EXECUTIVE SUMMARY

The goal of this milestone was to improve the high-order software ecosystem for CEED-enabled ECP applications by making progress on efficient matrix-free kernels targeting forthcoming ECP architectures. These kernels included matrix-free preconditioning and the development of new set of CEED solver bake-off problems.

As part of this milestone, we also released the next version of the CEED software stack, CEED-4.0, reported on results from several application collaborations, and documented the efforts of porting to AMD GPUs for Frontier and other modern architectures, such as Fugaku.

The specific tasks addressed in this milestone were:

- Port and run CEED benchmarks/miniapps on Frontier EA systems.
- Demonstrate performant libCEED integration in MFEM, Nek and applications.
- Matrix-free preconditioning of high-order operators.
- Benchmark problems for fast high-order solvers on GPU platforms.
- Public release of CEED-4.0.

The artifacts delivered include the next version of the CEED software stack, CEED-4.0, the next libCEED release, libCEED-0.8, and a number of developments integrated within applications to improve their GPU and CPU performance and capabilities. See the CEED website, `https://ceed.exascaleproject.org` and the CEED GitHub organization, `https://github.com/ceed` for more details.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The goal of this milestone was to improve the high-order software ecosystem for CEED-enabled ECP applications by making progress on efficient matrix-free kernels targeting forthcoming ECP architectures. These kernels included matrix-free preconditioning and the development of new set of CEED solver bake-off problems.

As part of this milestone, we also released the next version of the CEED software stack, CEED-4.0, reported on results from several application collaborations, and documented the efforts of porting to AMD GPUs for Frontier and other modern architectures, such as Fugaku.

The artifacts delivered include the next version of the CEED software stack, CEED-4.0, the next libCEED release, libCEED-0.8, and a number of developments integrated within applications to improve their GPU and CPU performance and capabilities. See the CEED website, `https://ceed.exascaleproject.org` and the CEED GitHub organization, `https://github.com/ceed` for more details.

2. MATRIX-FREE PRECONDITIONING OF HIGH-ORDER OPERATORS

Numerical solution of elliptic subproblems is a bottleneck in many multiphysics simulations. The challenge is often intrinsic to the stiffness of the underlying PDE where operators with fast timescale have been replaced by infinitely fast ones, which manifest as a steady-state elliptic problem at each timestep. The efficiency gains made with these models are generally worthwhile because they often lead to simpler, linear, subproblems that can be decoupled from other substeps (e.g., as in the case of incompressibility in the Navier–Stokes equations or the electric potential in the Poisson–Nernst–Planck equations), and can be preconditioned with well-understood techniques such as multigrid and multilevel Schwarz methods. Despite the fact that "fast" techniques are available for these problems, they often remain the most time-consuming step in the simulation, which makes the pursuit of faster techniques of continued interest, particularly for the Poisson problem.

A characteristic of the Poisson problem is that the Green's functions have long tails—distant information can have a nontrivial impact everywhere—problem is thus inherently communication intensive. Such problems are particularly challenging on GPU-based platforms where nonlocal communication costs are high compared with the rate of local work. In particular, efficient multilevel solvers require solution of coarse grid problems that have just a few degrees of freedom (dofs) per node and are communication dominated.

High-order discretizations for elliptic problems bring their own set of challenges. Per degree of freedom (dof), the condition number of the high-order stiffness matrix is higher than equivalent low-order discretizations. The stiffness matrix, if formed, is locally dense with $O(p^6)$ nonzeros per element for a pth-order FEM in 3D. Thus, for fast matrix-free formulations, the stiffness matrix is never formed and, and is therefore not available to be passed into black-box solvers such as algebraic multigrid (AMG).

There are, however, some structural advantages to high-order methods that lend themselves to preconditioning. These methods support a natural set of nested subspaces in that one can approximate the solution at lower polynomial order on the same mesh topology, which leads to p-multigrid (PMG), with the coarsest level corresponding to $p = 1$. For hex-based elements of order $p = 7$, the reduction in problem size is a factor of $p^3 = 343$, until one reaches the unstructured coarse grid problem ($p = 1$), which can then be addressed by AMG. A variety of smoothers can be considered, including alternating line [19], Schwarz [24], Chebyshev-Jacobi. In the case of Q_p elements (i.e., quads and hexes) an effective strategy is to construct a low-order finite element (FEM) approximation on the high-order nodal points. As first noted by Orszag [31], such an approach leads to bounded condition numbers, independent of (E, p), where E is the number of elements and p is the polynomial order. Consequently, the solution of a high-order problem is turned into one of solving a low-order (sparse) problem, albeit one with geometric challenges.

2.1 CEED solver bake-off problems

Because this problem is ubiquitous and continues to be challenging, the CEED project has initiated a new set of *bake-off problems* (BPs) in which developers are invited to bring forth their best ideas to solve large-scale high-order Poisson problems for a variety of challenging test cases and to share these with the scientific simulation community. These *bake-off problems for solvers* (BPSs) were designed with the following goals:

- Propose a new set of BPs that address the challenge of scalable linear system solves for high-order methods.
- The new BPs specify the problem (with emphasis on high-order discretizations) and convergence criteria, but not the preconditioning method.
- This activity is intended as a collaboration with the ECP ST solver teams, particularly *hypre* and PETSc.
- In addition to developing the specifications of the new BPs, the CEED team is also providing implementations and is working on optimizing their performance.
- This work involves solver algorithms in Nek, MFEM and libParanumal as well as GPU-specific solver optimizations, similar to the current BPs.
- The solver BPs will be documented on the CEED website.

The first task in designing the solver bake-off problem is to specify the problem geometry. It is of significant interest to understand the solver performance on curved and distorted geometries, with potentially high-aspect-ratio elements. For that reason, the bake-off problem is posed on a family of meshes, which we refer to as *Kershaw meshes* [21]. This family is parameterized by two anisotropy parameters $\varepsilon_y, \varepsilon_z \in (0, 1]$, such that the resulting mesh is completely uniform for $\varepsilon_y = \varepsilon_z = 1$, and the mesh becomes increasingly anisotropic as ε_y and ε_z decrease. The Kershaw mesh is obtained by deforming a uniform grid of the unit cube as follows. The x-axis is decomposed into 6 equally spaced layers. Elements with aspect ratios $1/\varepsilon_y$ and $1/\varepsilon_z$ are placed in the leftmost and rightmost layers, in opposing corners, and the intermediate layers are used to transition between these configurations in a Z pattern. The transition function may be piecewise linear "ramp" function ($s = 1$, where s denotes the polynomial degree of the transition), or it may be a smoothed version, either a C^1 cubic ($s = 3$) or a C^2 quintic ($s = 5$). Several different configurations of the Kershaw mesh are shown in Figure 1.

To date, we have preliminary results from several groups within CEED and an initial problem specification. The study was advertised at SIAM CSE in March, 2021, and we have been contacted by the deal.II group in Germany, who are interested in participating. The current problem specification is listed below.

- **Mesh setup**
  - straight-sided Kershaw mesh ($s = 1$, with distortion parameters $\varepsilon_y, \varepsilon_z$)
  - baseline: $\varepsilon_y = \varepsilon_z = 1$,
    hard: $\varepsilon_y = \varepsilon_z = 0.3$
  - staring with $6 \times 6 \times 6$ and increase by 6
- **Boundary conditions**
  - pure Dirichlet
  - must also support pure Neumann
- **Stopping criteria**
  - relative l_2 residual reduction: 10^{-8}
- **Plotting**
  - x-axis: increasing problem size (ndofs)
  - y-axis: 1) iterations 2) solve time
  - curves for varying order
- **CSV values to report**
  - code ID
  - preconditioner ID
  - machine ID
  - number of nodes
  - number of MPI ranks
  - n_x, n_y, n_z
  - solution polynomial degree
  - number of 1D quadrature points
  - ε_y, ε_z
  - ndofs (including Dirichlet boundary)
  - niter
  - initial and final residuals
  - error
  - t_{setup} (preconditioner setup)
  - t_{solve} (total iter time)

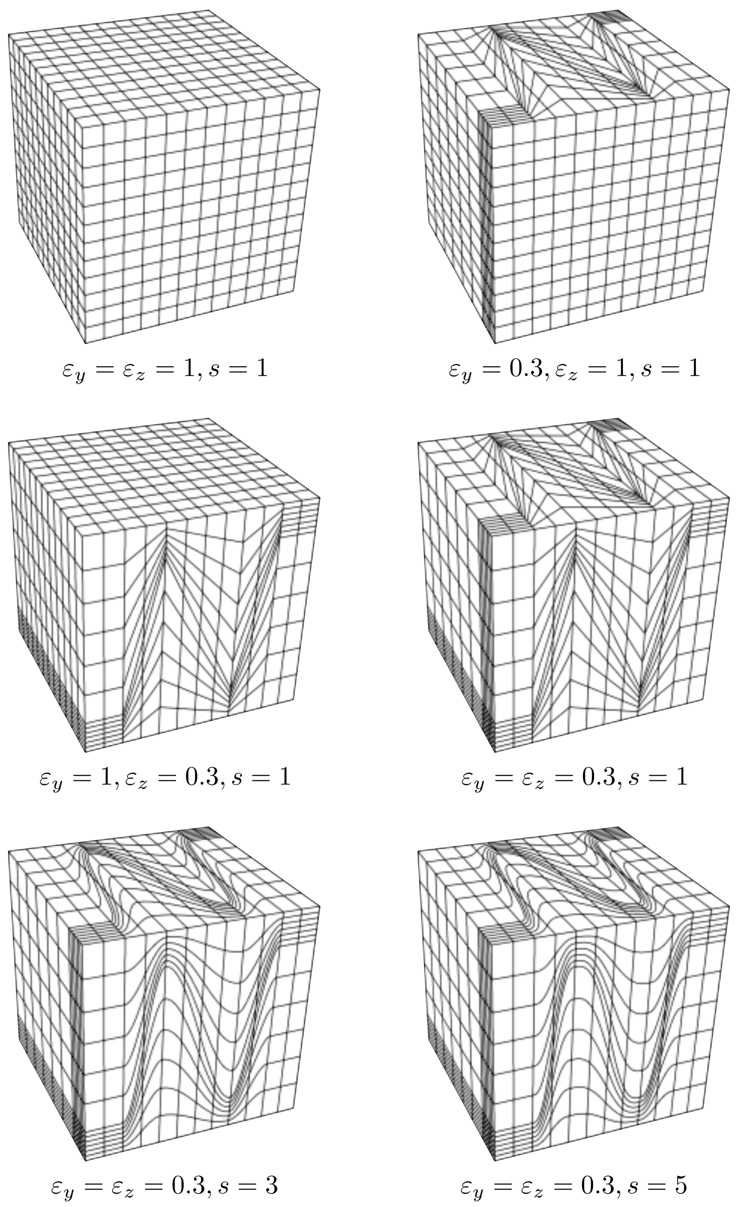

$\varepsilon_y = \varepsilon_z = 1, s = 1$
$\varepsilon_y = 0.3, \varepsilon_z = 1, s = 1$
$\varepsilon_y = 1, \varepsilon_z = 0.3, s = 1$
$\varepsilon_y = \varepsilon_z = 0.3, s = 1$
$\varepsilon_y = \varepsilon_z = 0.3, s = 3$
$\varepsilon_y = \varepsilon_z = 0.3, s = 5$

Figure 1: Configurations of the Kershaw mesh for different values of the anisotropy parameters ε_y and ε_z and degree s of the smoothed transition function.

We intend to also monitor the accuracy of the *numerical approximation* by prescribing a polychromatic right-hand side with known solution. It is important that such a solution be smooth so that spectral asymptotic convergence is possible, and yet contain multiple scales so the solution has genuinely global dependence on the forcing. For simplicity in implementing and varying boundary conditions, we also create a solution with zero Dirichlet and Neumann conditions, in which the continuous mean over the domain is zero. We consider the domain $x \in (-1, 1)$ and the primitive $s_k(x) = \sin(2k\pi x)$ in

$$u_k(x) = \exp(-1/s_k^2)\,\mathrm{sign}(s_k).$$

To achieve arbitrary solution complexity, we call $n \in \{1, 2, \dots\}$ the structure level and define the corresponding solution

$$w_n(x) = \sum_{j=0}^{n-1} u_{3^j}(x),$$

which we plot in Figure 2, alongside the second derivatives that appear in the corresponding forcing terms.

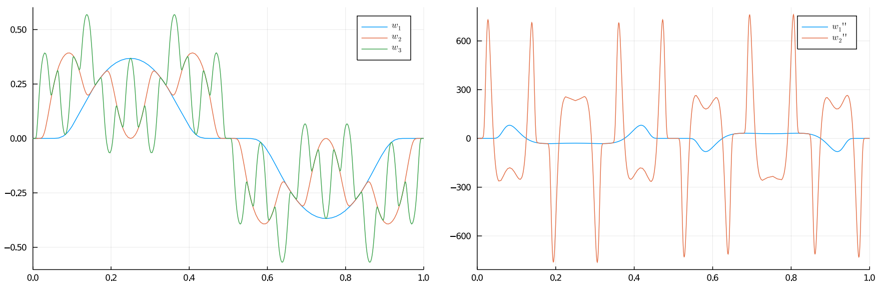


Figure 2: Solution $w_n(x)$ (left) and second derivatives $w_n''(x)$ (right) appearing the corresponding forcing terms at multiple structure levels n. Note that fine-scale features dominate the forcing terms at higher structure levels, but only generate solution features equivalent in size to the low frequencies.

This choice of solution requires all scales to be resolved equally to achieve a given accuracy, and high resolution can be demanded using higher structure levels n. It is extended to 3D via tensor product, $w_n(x)w_n(y)w_n(z)$. In order to facilitate comparisons of methods with different basis functions, we propose a convergence test based on a continuous (rather than discrete) norm of the error, which can be evaluated during a warm-up solve and used to define the discrete criteria needed during the timed solves.

We refer to the above problem as **CEED BPS3**, i.e. the solver version of the CEED BP3 bake-off problem. Similarly **CEED BPS5** refers to the CEED solver bake-off problem corresponding to the discretization for CEED BP5. In the following sections we present results from several of the CEED teams. The presentation and platforms are not yet uniform—these are preliminary results and we are still exploring the best approach to developing a robust problem specification that will be of maximal utility.

2.2 Initial BPS3 results with MFEM

Preliminary results using MFEM are available for the *BPS3* solver bake-off problem. We have developed a miniapp that solves the given Poisson problem on the Kershaw mesh, allowing the user to specify the problem parameters (mesh size, distortion, polynomial degree, etc.) and preconditioner in a flexible manner. The MFEM miniapp supports a variety of matrix-free preconditioners, built using a flexible combination of h-multigrid, p-multigrid, and low-order refined (LOR, or FEM–SEM) preconditioning. These preconditioning options are suitable both for the CPU and GPU. The coarsest level in the geometric multigrid hierarchy is approximately solved using an algebraic multigrid method. Both *hypre* and NVIDIA's AMGX algebraic multigrid methods are supported. Additionally, the user can control the parameters of the AMGX preconditioner using a JSON configuration file.

Results using two solver approaches on two variants of the Kershaw mesh are shown in Figure 3. The geometric hp-multigrid preconditioner gives the best performance for the case of the uniform grid ($\varepsilon_y = \varepsilon_z = 1$), but performance degrades significantly on the highly distorted mesh ($\varepsilon_y = \varepsilon_z = 0.3$). On the other hand, while the LOR preconditioner results in somewhat slower convergence on the uniform grid when compared with hp-MG, the iteration counts remain bounded and compare favorably with the hp-MG preconditioner on the distorted mesh case.

We also remark on the importance of the choice of quadrature rule used for the low-order refined discretization. In particular, significantly improved convergence can be obtained by using collocated Gauss–Lobatto quadrature on the low-order refined mesh (i.e. quadrature points at the mesh vertices), when compared with more accurate Gauss–Legendre quadrature. This corroborates the single-element results for spectral methods shown in [9]. Although both preconditioners give uniform convergence (independent of the polynomial degree) the constants are significantly improved using the collocated quadrature. The collocated quadrature has the additional advantage that the resulting system is both sparser and more amenable to AMG preconditioning, making the resulting LOR preconditioner more effective and less expensive to form. In Figure

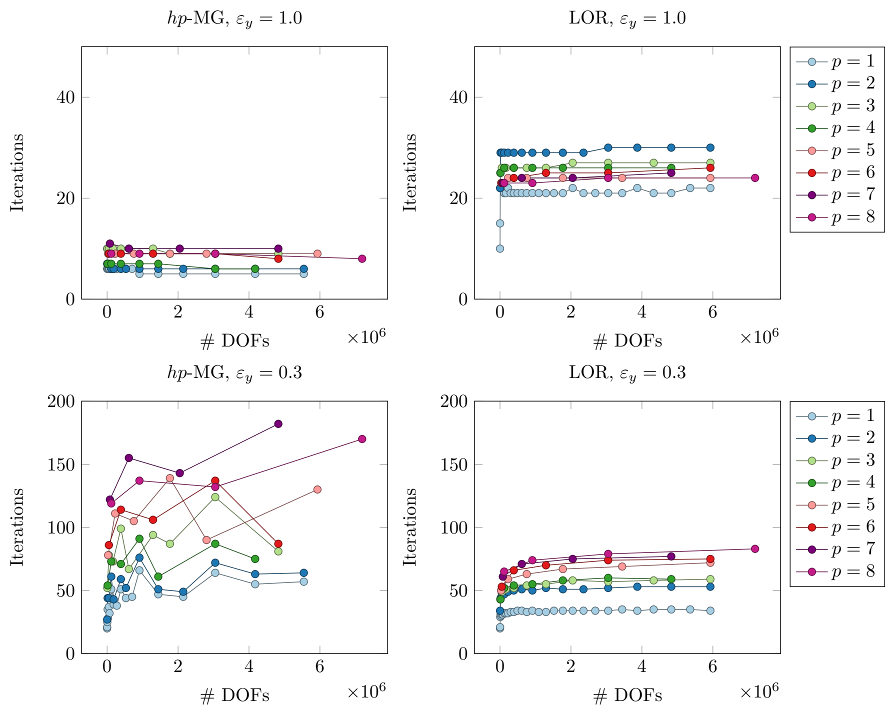


Figure 3: MFEM iteration counts for BPS3. Left column: hp-multigrid with Chebyshev smoothing and AMGX coarse solve. Right column: LOR preconditioning with AMGX V-cycle. Top row: uniform grid ($\varepsilon_y = 1.0$). Bottom row: highly distorted mesh ($\varepsilon_y = 0.3$).

4, the iteration counts using these two approaches are compared. We additionally consider the "one-per-vertex" tetrahedral LOR discretization proposed in [5]. For the BPS3 test case, this tetrahedral discretization results in identical sparsity pattern as the collocated quadrature, and almost identical preconditioning performance. However, as reported in [5], we expect the convergence to be significantly improved on cases with pure Neumann boundary conditions.

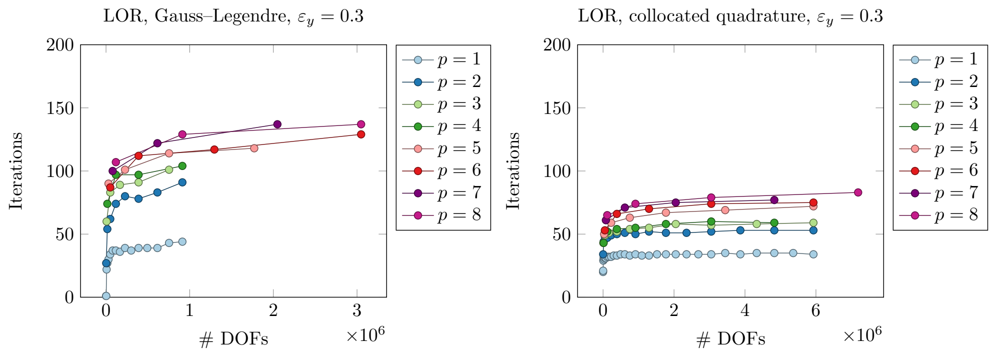


Figure 4: Comparison of iteration counts using LOR preconditioning with Gauss–Legendre quadrature (left) and collocated quadrature at vertices (right) for BPS3.

2.3 Initial BPS3 and BPS5 results with libParanumal

The CEED Virginia Tech team has upgraded the architecture of the *libParanumal* [10] linear solvers and the multigrid preconditioner hierarchy. Major updates including improved initial guesses when solving a sequence of linear systems [4]; switching storage of unknown from E-vectors to L-vectors (as used in [32]); and a comprehensive top-to-bottom overhaul and optimization of the algebraic multigrid, SEM-FEM, and p-multigrid preconditioning modules. These upgrades will be described in future publications.

For CEED BPS benchmarking we extended *libParanumal* to enable creation of Kershaw mapping of domains [21]. The exact solution was specified as a tensor product of sine functions.

When we analyzed the initial performance of the CEED BPS benchmarks we determined that comparing the performance of the solvers over a wide range of mesh sizes and polynomial degrees requires careful attention to the choice of PCG stopping criterion. Naively iterating the linear solver until the initial residual is reduced by eight digits may lead to biased results as lowest resolution solutions will likely be converged to many more significant digits than the leading order of approximation error. Thus when reporting solution times for low resolution the results will include unnecessary iterations that are not warranted given that the approximation error will swamp the iterative convergence error.

To illustrate this issue we implemented two iterative stopping criteria: firstly vanilla residual reduction of eight digits and secondly an a posteriori error indicator based condition that seeks to iterate until either the residual is reduced by eight digits or the numerical error dominates the iterative convergence error (see for instance Arioli [3] and the survey by Ainsworth and Oden [1] on a posteriori error estimates for finite element analysis). In the following we present results that demonstrate the a posteriori indicator approach preserves the leading term in the approximation error while reducing the iteration count and significantly decreases time to solution unless the residual reduction falls below the CEED target residual reduction. This did not require knowledge of the exact solution and achieves three goals: iteration reduction commensurate with desired numerical approximation; improved time to solution; balanced performance study over a range of mesh sizes and polynomial degrees.

The a posteriori indicator approach is particularly effective for the CEED BPS cases with piecewise cubic or quintic Kershaw map where the errors are relatively large due to the accuracy reduction caused by the nonlinear coordinate transforms. Of course the error indicator does incur extra operations however it typically does not increase the cost per iteration by more than 15% compared to using the standard CEED residual reduction.

***libParanumal* CEED BPS3.** This problem involves solving the Poisson problem on a uniform hexahedral mesh deformed via piecewise linear, cubic, and quintic Kershaw coordinate transforms with deformation parameters $\varepsilon_y = \varepsilon_z = 0.3$ [21]. The BPS3 variant pairs Lagrange elements of $(N+1)^3$ Gauss-Legendre-Lobatto nodes with a tensor-product Gauss-Legendre quadrature of size $(N+2)^3$.

In Figure 5 we show the H_1 convergence curves for a sequence of meshes. The charts show that the leading error for the residual and indicator based stopping criteria are in close agreement over a range of polynomial degrees and mesh resolutions for all three Kershaw maps. The same figure also shows that the PCG iteration counts are significantly smaller when using the indicator based stopping criterion than the residual based criterion. The iteration gap is more pronounced for the higher order Kershaw map and this corresponds to the loosening of stopping criterion because of the increased approximation error for the curvilinear maps.

The reduction in iteration counts with the relaxed stopping criterion yields an improved time to solution on a single NVIDIA V00 hosted by AWS as shown in Figure 6. We also see that the throughput is improved considerably despite the additional overhead of computing the error indicator.

In summary the original BPS3 fixed residual convergence criterion is prone to under stating the performance of the solvers at low resolution. Using the indicator based stopping criterion yielded a substantial speed up without including prior information about the solution. These speed ups are in addition to those obtained by our aggressive optimizations to the *libParanumal* capabilities.

***libParanumal* CEED BPS5.** In Figure 7 we show the impact of using the indicator based stopping criterion for the CEED BPS5 benchmark. The trends are largely the same as for CEED BPS3 and any differences are due to the choice of quadrature rules used in either test.

Likewise in Figure 8 we show that the indicator based stopping criterion also benefits the throughput of *libParanumal* for BPS5 while maintaining leading order accuracy.

2.4 Initial BPS5 results with Nek5000/RS

Here we present initial results for BPS5 (the diagonal-mass-matrix Poisson operator) implemented in Nek5000 and NekRS. Nek5000 supports several production preconditioning strategies as well as a choice between

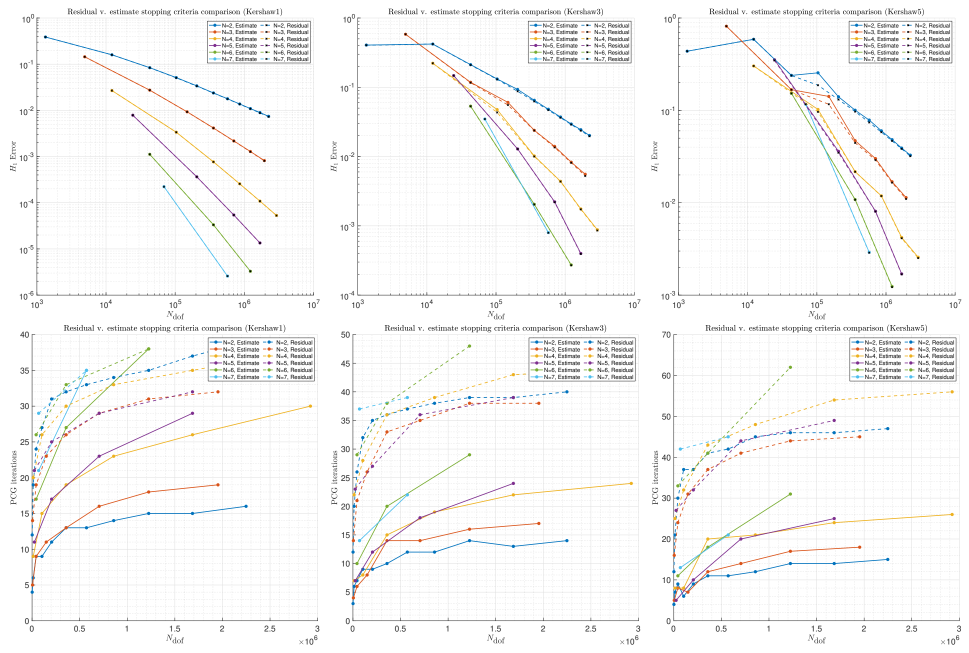


Figure 5: *libParanumal* CEED BPS3 ($\varepsilon_y = \varepsilon_z = 0.3$) benchmark results on a single NVIDIA V100 hosted by AWS. Top: comparison of approximation error when using the residual and indicator based PCG stopping criteria for BPS3 for the linear, cubic, and quintic Kershaw maps for a sequence of meshes at polynomial degrees $N = 1 : 7$. Bottom: comparison of iteration counts for the stopping criteria. Notice that the H_1 error is very similar for both stopping criteria and yet the iteration counts are smaller in general for the indicator based stopping criterion. This is particularly noticeable for the higher order Kershaw maps which induce larger numerical error allowing the indicator based stopping criterion to stop the PCG iterations even earlier.

FlexCG and GMRES. The principal production preconditioners for Nek5000 are p-multigrid (PMG) with (strictly) additive Schwarz preconditioning (ASM) [15, 24] and FEM-based preconditioning [5] that uses low-order approximations, as first proposed by Orszag in the early 80s [31]. NekRS supports PMG with ASM smoothing, Chebyshev-accelerated Jacobi smoothing (Cheb-Jac), or Chebyshev-Schwarz smoothing that is multiplicative between levels (Cheb-Schwarz).

Nek5000 Iteration Counts on Monza CPU. Below are SEM results with Nek5000 for the Kershaw problem with $\varepsilon = 1$ and 0.3.

- All of these cases use GMRES-50 as we want to understand the preconditioner performance in the context of (near) perfect projection before adding variables such as FlexCG, etc.
- For modest element counts ($E < 10^6$), we have the option of using the direct XX^T-base coarse-grid solver or AMG. For larger values of E, AMG is generally faster. Nek5000 uses the gslib version developed by James Lottes, or some combination of that with Hypre doing the setup work.
- For the "easy" case of $\varepsilon = 1$, the ASM case is the fastest as it has a lower iteration count than FEM-SEM and a significantly lower cost per iteration [5].
- Although timing results are not shown here, for $\varepsilon = .3$, the Hypre-based FEM-SEM solver is the fastest.
- The "direct" FEM-SEM preconditioner is realized by running 10 V-cycles of Hypre for each FEM solve and is not competitive from a timing standpoint. It is used simply to establish a lower bound on iteration count.

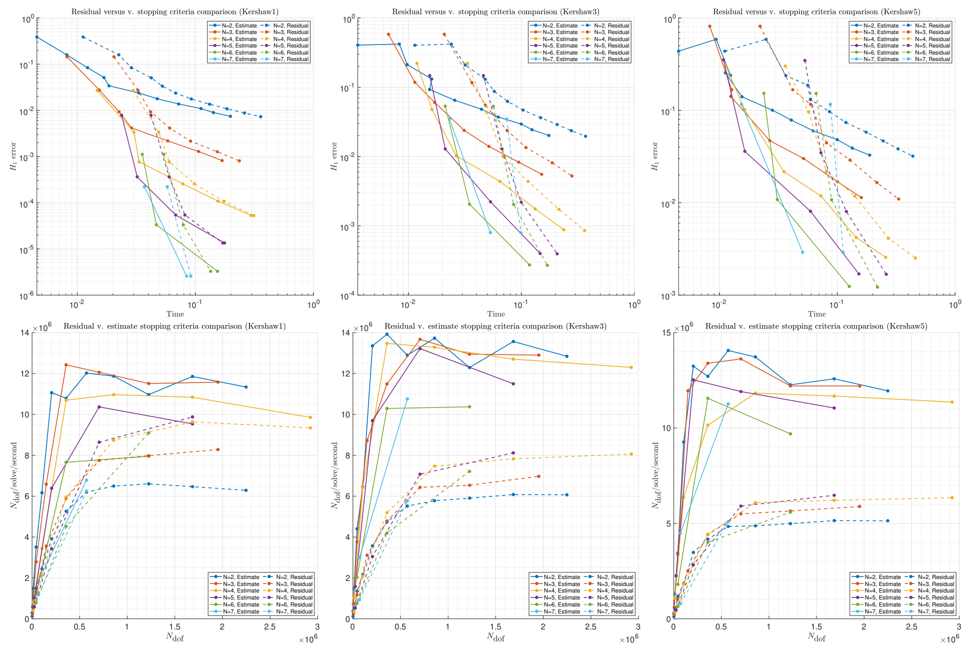


Figure 6: *libParanumal* CEED BPS3 ($\varepsilon_y = \varepsilon_z = 0.3$) benchmark results on a single NVIDIA V100 hosted by AWS. Top: comparison of H_1 error versus solver time for residual and indicator based PCG stopping criteria. Bottom: comparison of solver throughput.

- We note that GMRES is critical to getting reasonable iteration counts with FEM-SEM.

Nek5000/NekRS Results on Summit CPU. Here we make CPU-based comparisons of the Nek5000/RS preconditioners. NekRS has several choices for PMG smoothers but we focus on two of the usual choices, Cheb-Schwarz and Cheb-Jac. We see that, for $\varepsilon = 0.3$, FEM-SEM on Nek5000 is outperforming the PMG approach both in iteration count and time. Regarding iteration count, we would expect this result given that FEM-SEM tends to be very robust for challenging problems (which is certainly the correct characterization of the Kershaw $\varepsilon = 0.3$ case). Unfortunately, we do not yet have FEM-SEM working in NekRS as it is under development, so we cannot yet make a meaningful comparison on Summit's V100s.

3. PERFORMANCE IMPROVEMENTS FOR GPUS AND ARM

In this section we review recent performance improvements for AMD GPUs and report on our initial experience with porting to the Fugaku system and its Fujitsu A64FX ARM chips.

3.1 New HIP backends for libCEED

In the previous milestone report [22], we reported on the development of the first two HIP backends for libCEED: the `hip-ref` backend, ported from `cuda-ref` and providing the foundation for `hiprtc` usage across future HIP backends; and a HIP version of the MAGMA backend. Progress has continued on porting libCEED's CUDA capabilities to HIP. Two new HIP backends have been added to libCEED: `hip-shared` and `hip-gen`, corresponding to `cuda-shared` and `cuda-gen`, respectively. The *shared* backends are so-named for their use of shared memory to optimize the basis kernels; the *gen* backends use code generation to create one fused kernel for the entire libCEED Operator application, and are expected to achieve the best performance for tensor-basis Operators (for `cuda-gen`, this has been demonstrated in previous reports). The current performance of the new backends for the tensor-basis diffusion benchmark (BP3) on an AMD MI50 GPU is shown in Figure 11, as well as the performance of the deterministic hipMAGMA backend (`gpu/hip/magma/det`) and MFEM's default HIP backend. The libCEED backends use runtime compilation with `hiprtc`, while

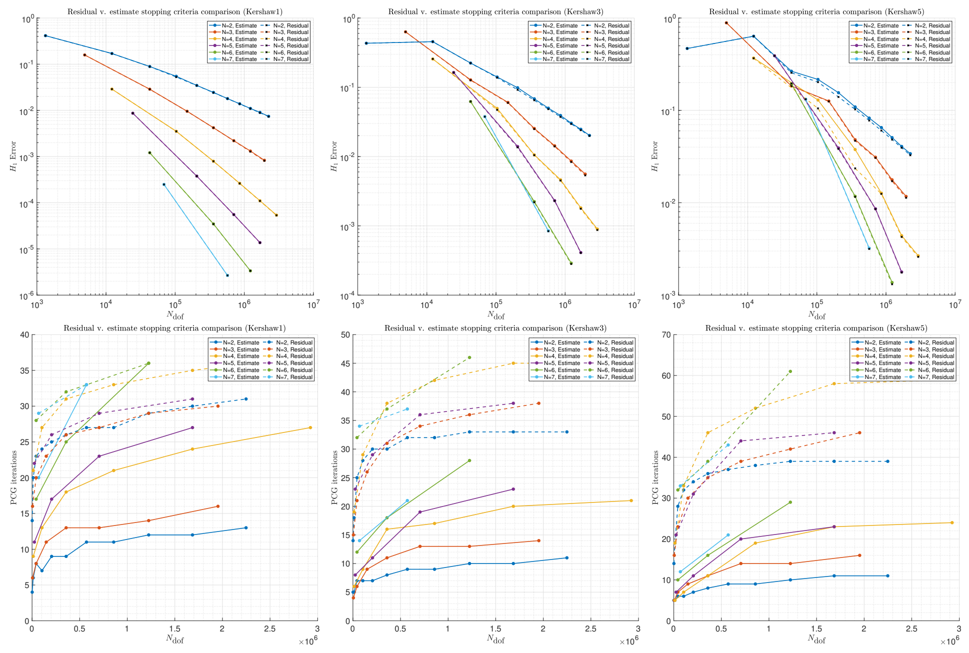


Figure 7: *libParanumal* CEED BPS5 ($\varepsilon_y = \varepsilon_z = 0.3$) benchmark results on a single NVIDIA V100 hosted by AWS. Top: comparison of approximation error when using the residual and indicator based PCG stopping criteria for BPS5 for the linear, cubic, and quintic Kershaw maps for a sequence of meshes at polynomial degrees $N = 1 : 7$. Bottom: comparison of iteration counts for the stopping criteria. Notice that the H_1 error is very similar for both stopping criteria and yet the iteration counts are smaller in general for the indicator based stopping criterion. This is particularly noticeable for the higher order Kershaw maps which induce larger numerical error allowing the indicator based stopping criterion to stop the PCG iterations even earlier.

MAGMA and MFEM use templated kernels for different orders of basis functions. MFEM results are missing eighth order basis functions due to a runtime error for those kernels.

As before, the development of the new HIP backends began with a *hipification* of the CUDA code. However, in the case of the `hip-shared` backend, the performance was unexpectedly poor when compared to the reference backend, `hip-ref`. Profiling the two backends during a run of the MFEM diffusion benchmark problem indicated one source of the performance degradation was the use of constant memory. Specifically, the `cuda-shared` backend copies arrays associated with the basis to constant memory prior to basis kernel application, using `cudaMemcpyToSymbol`. Figure 12 shows a snapshot of the profiler output for the reference and the *originally ported* shared backend, highlighting the gap between the end of the QFunction kernel and the beginning of the second basis kernel (gradient in this case, marked as *grad*). We see that not only is the gap larger for the shared backend than for the reference backend, the time during the execution gap in the shared case is dominated by calls to `hipMemcpyToSymbol`. HIP on AMD hardware treats constant memory differently than CUDA, and the negative result on performance is evident.

Therefore, in the `hip-shared` backend, the use of constant memory was replaced with device functions that read the basis arrays in question into shared memory. These changes, included in what is now the current `hip-shared` backend, led to improvements of between 30-80% on an AMD MI50 GPU when compared to the direct port, as seen in Figure 13.

In the case of the ported `hip-gen` backend, one concern was the current status of the atomic operations for AMD hardware; the `cuda-gen` backend requires `atomicAdd` to ensure correct results. To obtain an initial rough estimate for how the use of atomic operations affects benchmark performance for AMD/HIP compared to NVIDIA/CUDA, we compare the results of the MFEM diffusion benchmarks for the `cuda-gen` and `hip-gen` backends to a version of each which removes the `atomicAdd` calls. Of course, these modified no-atomics

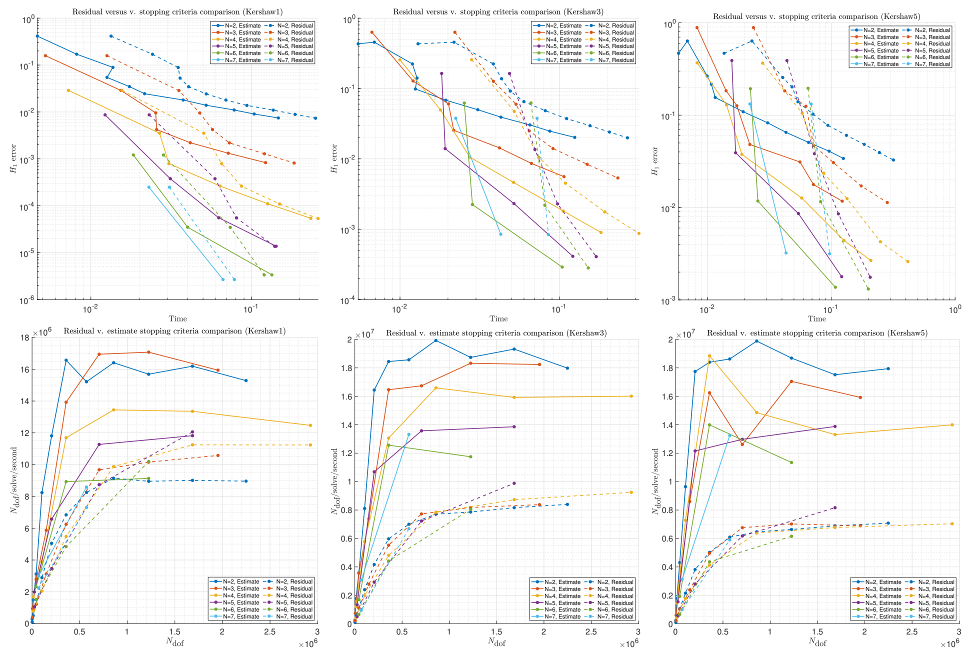


Figure 8: *libParanumal* CEED BPS5 ($\varepsilon_y = \varepsilon_z = 0.3$) benchmark results on a single NVIDIA V100 hosted by AWS. Top: comparison of H_1 error versus solver time for residual and indicator based PCG stopping criteria. Bottom: comparison of solver throughput.

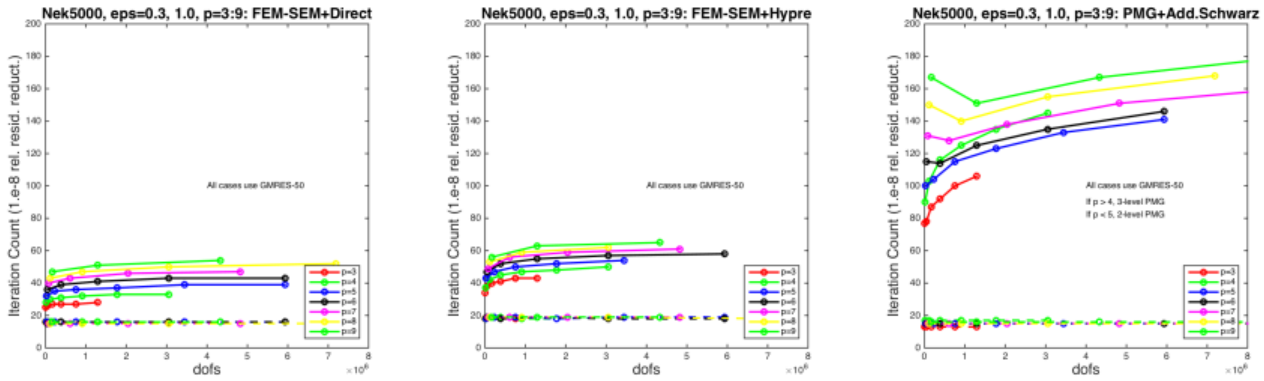


Figure 9: Nek5000 results for Kershaw 3D test with $\varepsilon = 1$ (dashed lines) and 0.3 (solid lines) for $p = 3 : 9$ and $E = 6^3, 12^3, \ldots, 36^3$. (Some larger cases did not run because of lack of memory on the 8-core Monza.)

versions will likely not give the correct results, and cannot be considered a valid modification of the backends for actual use; we aim only to gain an understanding of how requiring an atomic operation could affect the performance for the different GPUs. Figure 14 shows the ratio of the benchmark performance metric (DOFs $\times$ CG iterations / time) for the case with no atomics compared to the correct backend code. For the largest problems, where the GPU is most efficient, the current behavior is clearly different for HIP on the MI50 than for CUDA on the V100. For `cuda-gen`, any cost of the atomic addition is offset by the costs of any thread interference without it, and the atomic version is faster; for `hip-gen`, the atomic operation is always more costly after the total number of degrees of freedom (DOFs) exceeds 3×10^5. However, this effect is relatively small for the larger basis functions orders in large problem sizes, and it is not sufficient to explain the lack of performance of `hip-gen` compared to `hip-shared` in Figure 11 – we expect to be able to achieve much better maximum performance for the fused kernel of `hip-gen` compared to the non-fused kernels of `hip-shared`.

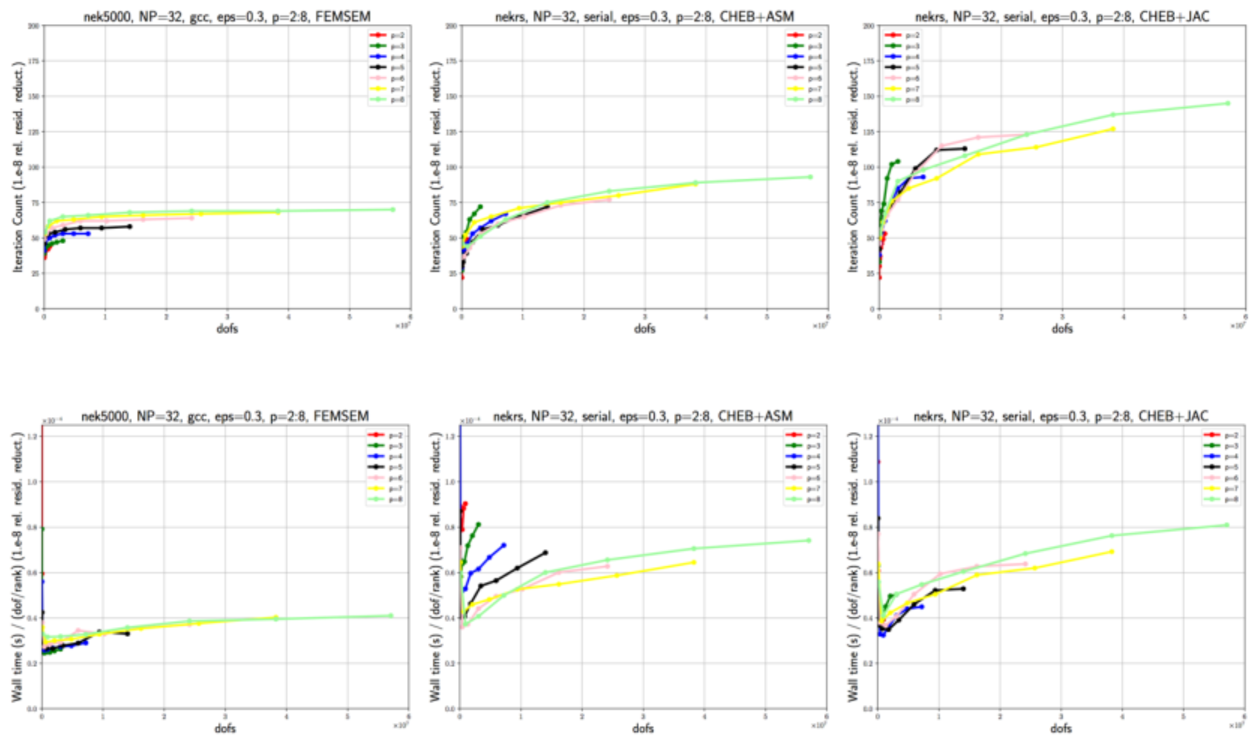


Figure 10: Nek5000/RS preconditioning results on Summit CPUs: FEM-SEM (left), Chebyshev-Schwarz PMG (center), and Chebyshev-Jacobi PMG (right): iteration counts (top) and time-per-dof (bottom).

3.2 Nek performance on Fugaku vs. Summit

The Nek5000 proposal was selected to perform the half-system scale simulation during the period of March 16–18 on Fugaku. In total, seven teams were selected and each team was given *PHASE I* and *PHASE II* for half-system scale runs with four hours in each phase.

Our test approach for Fugaku is the same that we follow for any new machine. We begin with an assessment of single-core performance and internode communication overhead, with a particular emphasis on short-messages relevant to strong-scale performance, as detailed in [13]. Following these unit tests, we perform strong- and weak-scale studies for full Navier-Stokes simulations. The full list of test cases on Fugaku is shown in Table 1, demonstrating ECP CEED applications in collaboration with ECP ExaSMR, ECP ExaWind and DOE NEAMS teams. We have performed these tests on other platforms, using CPUs and GPUs on up to full-system scale of Summit, so that we can compare to results from Fugaku.

We initiated with ping-pong test for the system analysis, shown in Figure 15 and `mxm` performance measurement, shown in Table 2. We performed small size bake-off problems (BP5) in Figure 16 and turbpipe flow simulations that are in the range of 10–128 nodes in Table 3.

The small- and medium-size problems using 100–1272 nodes ran without difficulty once we sorted out the requisite compiler flags for Nek5000. For the problem sizes using larger than 2000 nodes, however, we observed limitations with dramatic increase in timings for MPIIO while reading the input mesh. During the *PHASE I* – 4 hours time slot, we submitted all of our large scale jobs for an hour in the range of 2000–82944 nodes and we observed that all of them got stuck in reading mesh files for cases where the input mesh files are of modest (5.4GB) to large size (78 GB in the largest cases). We had also prepared restart files that are as big as 1.4TB, but with the I/O difficulties we started preparing the cases that do not require restart files. We had 24 hours before *PHASE II*, which was insufficient time to implement and test for non MPIIO ready, we tried ROMIO option in job submission script, focusing on the test cases without requirement of restart file. The ROMIO didn't make difference and all the big jobs still got stuck reading the input file for

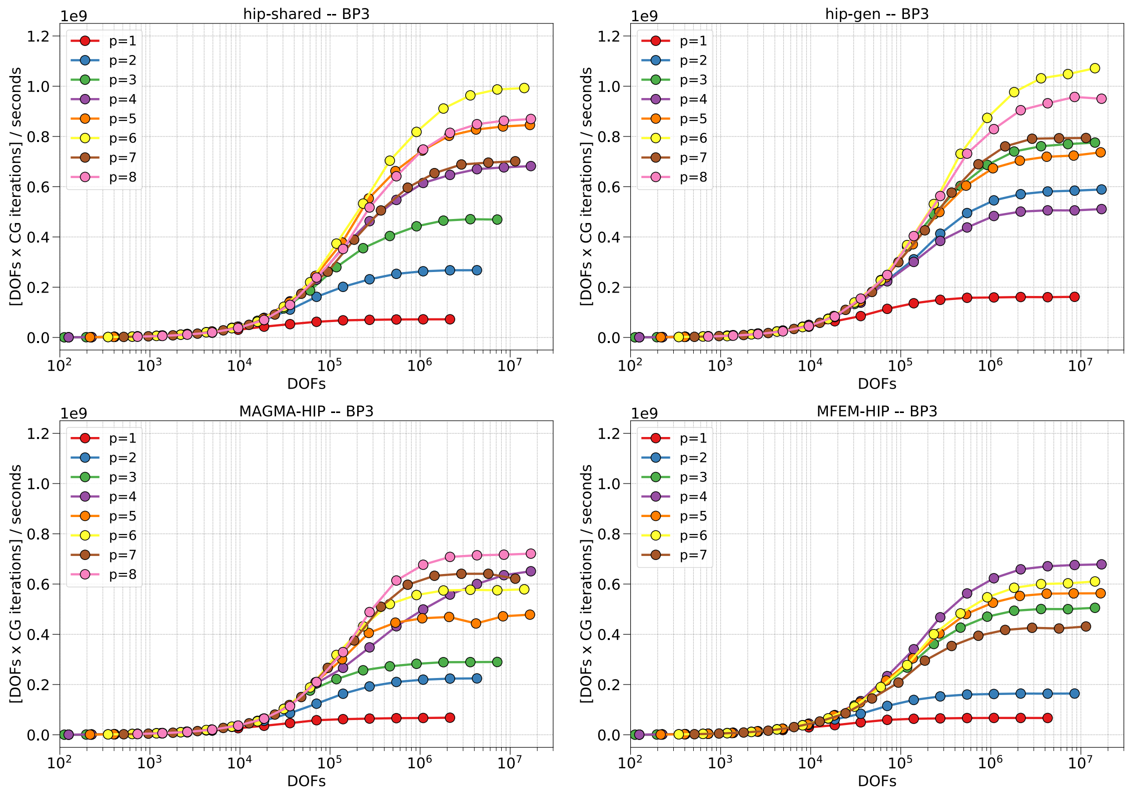


Figure 11: Performance comparison of the tensor-basis scalar diffusion benchmark (BP3) for four available HIP backends in MFEM: libCEED `hip-shared`, top left; libCEED `hip-gen`, top right; libCEED hipMAGMA (deterministic), bottom left; default MFEM HIP backend, bottom right. Results obtained on an AMD MI50 GPU with ROCm 4.0.

CEED apps	Test Case on Fugaku	abbreviation	scaling	gridpoints	node size	size
CEED	Small dgemm (90% of Nek5000 ops)	mxm		0.5M	1	small
	System latency tests with pingpong	pingpong		0.5M	10	
	CG Poisson solve	BP5	strong	11M	4–128	
	turbulent pipe flow	turbpipe	strong	0.5M	3–24	
ExaWind	Atmospheric boundary layer flow	ABL	strong	89M	128–1024	medium
ExaSMR	17×17 rod bundle	rod1717	weak	95M–60B	106–46640	
ExaSMR	17×17 rod bundle	rod1717	strong	60B	19872–82944	large
NEAMS	Full-core reactor, 352625 pebbles	350k pebble	strong	33B	19872–82944	

Table 1: Nek5000 Tests Cases on Fugaku.

the meshes (100MB–78GB). Fugaku half-system scale simulation support team informed their limitation of MPIIO support on Fugaku and they are currently looking into this problem how this can be improved on the system side.

In this report, we demonstrate what problem sizes and applications we have run and what solutions we are preparing to move forward on Fugaku. To move further, on Nek side, we wrote a new parallel I/O routine that can specifically support runs on Fugaku. We expect that with our new parallel I/O update, not using MPIIO, combined with Fugaku specific `llio_transfer` support, we would be able to move forward the larger case above 2000 nodes. These new implementation are currently under testing on Summit at scale while Fugaku is under maintenance period and not available to run large scale on Fugaku.

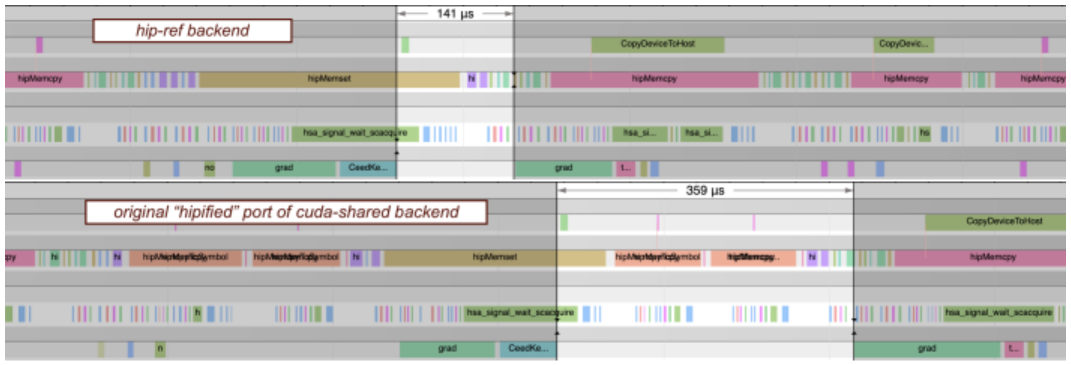


Figure 12: Snapshot from profiling results for the diffusion benchmark (BP3) with third-order basis functions. The highlighted gaps show the time between finishing the QFunction kernel and launching the second basis action (gradient) kernel. Top: `hip-ref` backend. Bottom: original `hip-shared` port, with same memory usage calls as `cuda-shared`. Results obtained on an AMD MI50 GPU with ROCm 3.9.

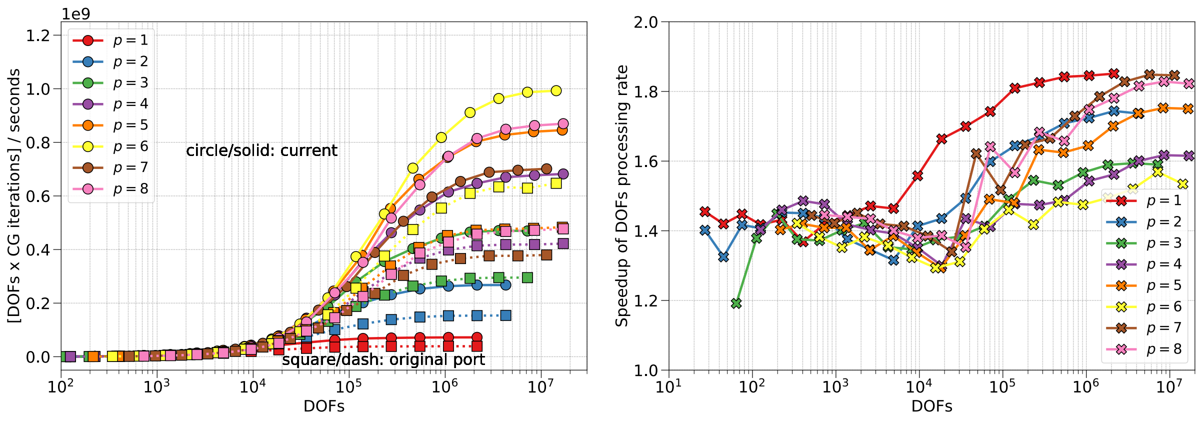


Figure 13: Comparison of libCEED's `hip-shared` backend with its original ported version for the diffusion benchmark (BP3) implemented with MFEM. Left: the rate of DOFs processed inside the CG solver. Right: speedup of the modified backend over the original port. Results obtained on an AMD MI50 GPU with ROCm 4.0.

Baseline Tests. We have made the following measurements:

- matrix-matrix product (mxm) timings, which reflect $>90\%$ of the flops in Nek5000
- ping-pong tests to gather baseline inter-rank latency and bandwidth
- all-reduce tests
- Poisson solve scaling out to 4096 ranks, using 128 nodes. This is the CEED bake-off problem, *BP5*, extensively analyzed for strong scaling in [14].

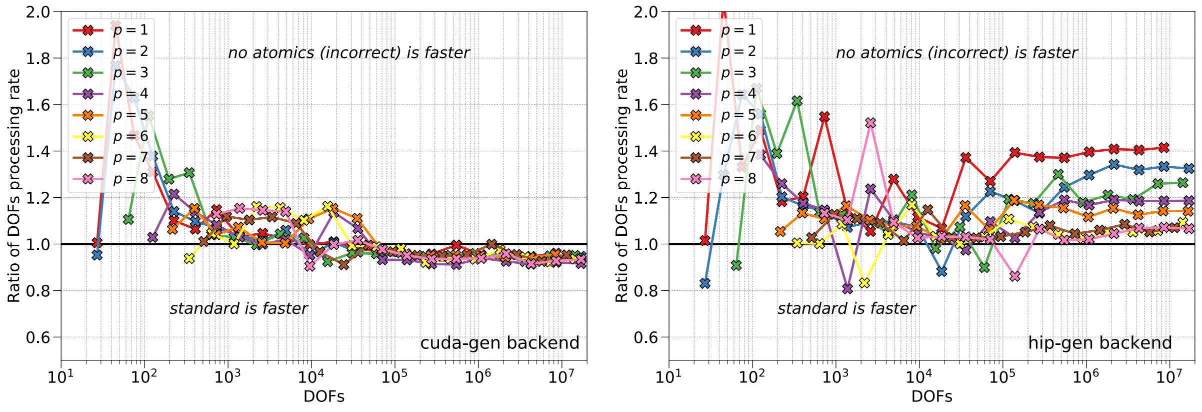


Figure 14: Comparing the effects of `atomicAdd`s in (left) `cuda-gen` and (right) `hip-gen` for the diffusion benchmark (BP3). Results obtained on an NVIDIA V100 GPU with CUDA 11.1 and an AMD MI50 GPU with ROCm 3.9.

code	Inc	IInc	IIInc	Ic	IIc	IIIc
mxm	6964	4076	4396	10507	4243	4846
mxm44	2143	2900	3432	3202	3055	3739
mxmd	3643	641	656	4343	646	672
mxmu2	6194	3978	4345	10347	4183	4773
mxmu3	6450	3379	701	9402	3719	720
mxmu4	639	152	153	669	155	155
mxmf3	6515	3979	4373	10312	4113	4823
mxmfb	6586	3968	4363	10313	4122	4826

Table 2: Nek5000: Matrix-matrix product performance (MFLOPS) on a single core of Fugaku: $C = AB$ with $A = m \times 8$, $B = 8 \times n$. Columns correspond to I: $m = 64$, $n = 8$, II: $m = 8$, $n = 8$, III: $m = 8$, $n = 64$. The nc and c flags indicate whether the matrices were cached or not cached during the timings.

- Strong scaling for full Navier-Stokes out to $P = 128$ ranks for turbulent flow in a pipe.

We started with Nek5000's *platform_timer* utility which runs a battery of matrix-matrix (`mxm`) product timings for sizes relevant to the spectral element method (SEM). For spectral element polynomial expansions of order N matrix sizes are I: $(M \times M) \times (M \times M^2)$, II: $(M \times M) \times (M \times M)$, and III: $(M^2 \times M) \times (M \times M)$, where $M = N + 1$ is the number of Gauss-Lobatto-Legendre (GLL) points used in each direction within each spectral element. These mxms account for $> 90\%$ of the flops Nek5000, so this is the *performance-critical kernel.* As indicated in Table 2, the tests are run both with cached and non-cached data. In Nek5000, the data is neither completely in or out of cache so we usually average these results when estimating likely flop rates. Nek5000 defaults to the `mxm` case in the table (top row), and we see that a single core is getting about 4 to 10 GFLOPS for $M = N + 1 = 8$ (i.e., N=7th-order polynomials that are typical of production runs). The performance is about 10% of the Fugaku Linpack peak, which is reasonable given the relatively small matrices used in Nek5000. We also note that these values are about 4 to 7 times faster than what we typically sustained on BG/Q for the same tests.

The *platform_timer* also provides message-passing timing info by running a sequence of ping-pong tests from rank 0 to rank p, $p = 1, 2, 3, \ldots, 511$, for a number of 64-bit words ranging from $m = 1$ to 10^5. Figure 15 shows the results of this study (with the on-node ranks removed for clarity). In our initial trials for Fugaku, we see two distinct curves: one for ranks $p < 185$ and one for $p > 185$, where the latter one has significant inter-rank latency for messages of size $m > 32$ words. This value of m is important because many of the messages during the SEM gather-scatter data exchange are of size m =64 (for the case $N = 7$), which means

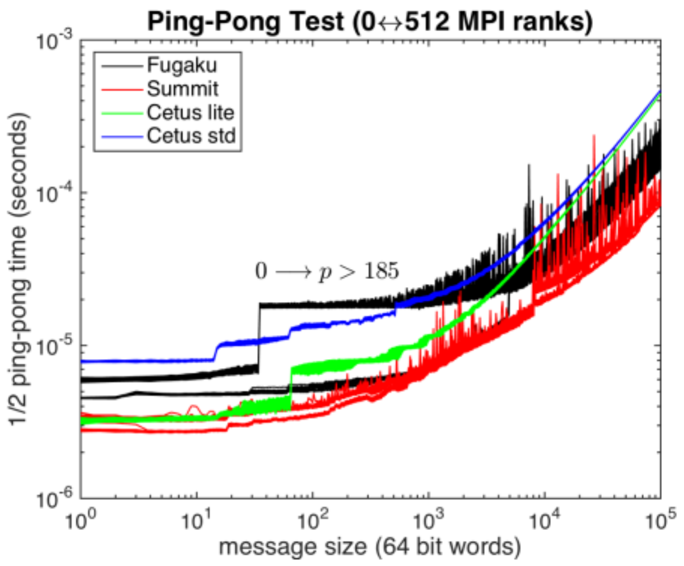


Figure 15: Ping-pong tests on Fugaku, ALCF Blue Gene/Q (Cetus), and OLCF Summit.

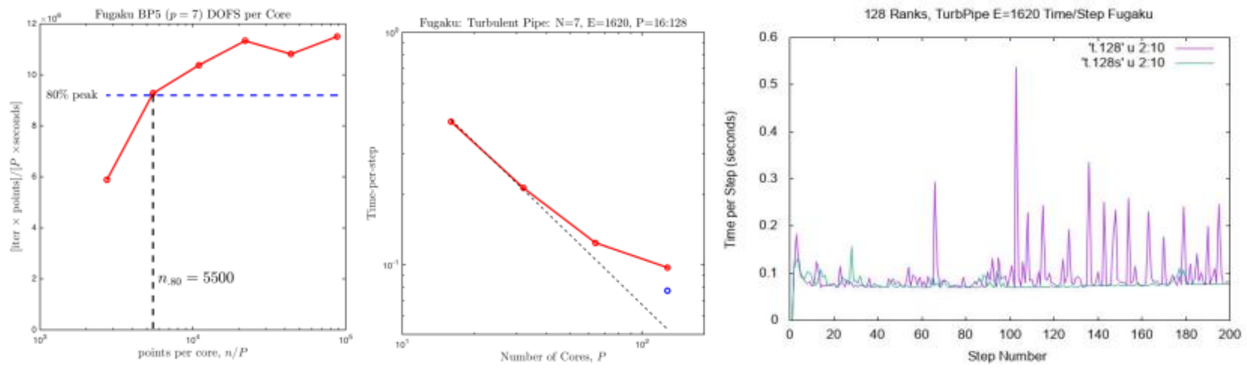


Figure 16: Fugaku scaling results: (left) Poisson strong-scaling results for $P = 128 : 4096$ with 11M points; (center) turbulent pipe flow for $P = 16 : 128$, 0.5M points; (right) time per step for successive pipe flow runs at P=128.

that they incur a relatively high cost.

From the mxm and ping-pong measurements, we anticipate that $n_{0.8}$ (the number of gridpoints per rank to realize 80% efficiency) on Fugaku would be larger than on Mira because its cores are significantly faster but its latency is higher. Specifically, we can estimate that Fugaku should have about a $4 \times 1.5 = 6$-fold increase over the value of $n_{0.8} = 4000$ on Mira. That is, we expect the strong-scale limit (80%) efficiency on Fugaku to be about 24000 points per core and that is therefore (roughly) the size problem where attention to optimization should be focused.

Figure 16(left) shows timing results for the Poisson model problem *BP5* of [14]. This is a simple strong-scale study of Jacobi-preconditioned conjugate gradient iteration, run on P=128 to 4096 cores. Here, we find that the 80% efficiency mark $n_{0.8} \approx 5500$ points per rank. Figure 16(center) shows strong-scale results for a small Navier-Stokes simulation of turbulent pipe flow with E=1620 elements of order N=7. In this case, $n_{0.8} \approx 8700$. We note that the run time at P=128 varied by 25% between successive runs. The time per step, plotted in Fig. 16(right), shows significant spikes on several of the timesteps, some being 5× larger than the average. We assume that this is due to some type of network interference given that the same code was run in both cases.

Table 3 shows another strong-scale study at small scale for turbulent pipe flows with $E = 7840$, $N = 7$,

$n = 2.6M$ and $Re = 19000$, demonstrating $n_{0.8} \approx 5252$ points per rank.

Nek5000 on Fugaku Strong, Turbpipe Flows, $E = 7840$, $N = 7$, $n = 2689120$								
node	core	E	n	n/core	t_{step}	R	R_{ideal}	P_{eff}
3	128	7840	2.6M	2.1009e+04	0.77	1.00	1	100
6	256	7840	2.6M	1.0504e+04	0.41	1.87	2	93
12	512	7840	2.6M	0.5252e+04	0.24	3.20	4	80
24	1024	7840	2.6M	0.2626e+04	0.17	4.52	8	56

Table 3: CEED: Nek5000 Turbulent pipe flows on Fugaku. $E = 7840$, $N = 7$.

Performance at Medium Scale. We were able to conduct strong- and weak-scale studies on Fugaku at medium scale. (Unfortunately, MPIIO failed reading our meshes for large-scale problems, which is an issue that we are close to resolving by writing a stand-alone parallel reader since last week.) The strong-scale study case is the turbulent atmospheric boundary layer (ABL) problem for the DOE ECP project, ExaWind, and the weak-scale study is the 17×17 rod-bundle flow for the ECP ExaSMR project.

For the ABL flows, Table 4 demonstrates Nek5000 strong-scale results on Fugaku in comparison to NekRS GPU strong-scale performance on OLCF/Summit. The velocity solves use Jacobi-PCG with tolerance 1.e-06. For the pressure solve, an overlapping additive Schwarz method (ASM) and a Chebyshev-accelerated overlapping additive Schwarz method (CHEBY+ASM) smoothers are used for Nek5000 and NekRS, respectively, with spectral element multigrid preconditioner. HYPRE is used for coarse grid solve. We used characteristics-based BDF2 with 1 substep and applied projection in time for the pressure. We also show the average velocity (v_i) and pressure (p_i) iteration counts over the same simulation interval. The computational domain in nondimension is fixed with $\Omega = [0, 4]^3$ and we used $E = 64^3 = 262144$, $N = 7$, $n = 89M$, and $Re = 50000$. Restarting with initial condition at time= 6 hours, performed for 200 steps and the averaged timing per step, t_{step}, is measured in seconds for 101–200 steps with the timestep size Δt in nondimension and Δt^* in physical unit. The real time ratio is $r_t = t_{step}/\Delta t^*$ where $\Delta t^* = (L/u)\Delta t$ with the characteristic length, L=100m, and the stream velocity, 8m/s. We observe t_{step} on Fugaku using 32768 cores is similar to the one on Summit using 48 GPUs, demonstrating the real time ratio $r_t \sim 1.5$. In Table 5(top), we show the same case study, focusing on the increase of the MPIIO mesh read time as the number of cores increases, which will be discussed further in the next section, relating to the MPIIO issue.

For the 17×17 rods case, Table 5(bottom) demonstrates Nek5000 weak-scale results on Fugaku. We measured the average wall time per step in seconds, t_{step}, using 101-200 steps for simulations with $Re_D = 5000$. The approximation order is $N = 7$, and dealiasing is used with $N_q = 9$. We use projection in time, CHEBY+ASM, and flexible PCG for the pressure solves with tolerance 1.e-04. The velocity solves use Jacobi-PCG with tolerance 1.e-06. BDF3+EXT3 is used for timestepping with Δt= 3.0e-04, corresponding to CFL=0.54 for 17×17 rods case. The geometries for the weak-scaling studies were generated by extruding layers of 2D elements in the axial flow direction. For strong scaling of the 17×17 rods case, we used E =175M, totaling n =60 billion grid points.

For the 17×17 rods case, the pressure iteration counts, $p_i \sim 2$, are lower for these cases than for the pebble cases, which have $p_i \sim 8$ for the same timestepper and preconditioner. The geometric complexity of the rod bundles is relatively mild compared to the pebble beds. Moreover, the synthetic initial condition does not quickly transition to full turbulence. We expect more pressure iterations in the rod case (e.g., $p_i \sim$ 4–8) once turbulent flow is established.

Performance at Half-System Scale. We have made the following strong-scale measurements on Fugaku, to be compared to the results on OLCF/Summit:

- A long 17×17 rod bundle with a mesh using 175 million element, 60 billion grid points, and I/O Issue
- Full-Core 350K pebbles with a mesh using 98 million elements, 33 billion grid points, and I/O Issue

We have large case files for rod1717 and peb350k that we had I/O issue in reading our input mesh file. The file sizes and problem sizes are in Tables 6–10.

Nek5000 Fugaku Strong, ABL Flows, $E = 64^3$, $N = 7$, $n = 89,915,392$, Δt=5.e-3, Δt^*=6.25e-2, CFL=3.52, $\Omega = [0,4]^3$												
node	core	E	E/core	n/core	v_i	p_i	T_i	t_{step}	R	R$_{\text{ideal}}$	P$_{\text{eff}}$	r_t
128	4096	2.6214e+05	64	21952	3	4.5	1	3.05e-01	1.00	1	100	4.88
256	8192	2.6214e+05	32	10976	3	4.5	1	2.13e-01	1.43	2	71	3.40
512	16384	2.6214e+05	16	5488	3	4.5	1	1.34e-01	2.27	4	56	2.14
1024	32768	2.6214e+05	8	2744	3	4.5	1	9.48e-02	3.21	8	40	1.51

NekRS Summit Strong, ABL Flows, $E = 64^3$, $N = 7$, $n = 89,915,392$, Δt=5.e-3, Δt^*=6.25e-2, CFL=1.89, $\Omega = [0,4]^3$												
node	gpu	E	E/gpu	n/gpu	v_i	p_i	T_i	t_{step}	R	R$_{\text{ideal}}$	P$_{\text{eff}}$	r_t
2	16	2.6214e+05	1.6384e+04	5.6197e+06	1.7	1.86	1	2.19e-01	1.00	1.00	100	3.51
3	18	2.6214e+05	1.4564e+04	4.9953e+06	1.7	1.86	1	2.01e-01	1.08	1.12	96	3.22
4	24	2.6214e+05	1.0923e+04	3.7465e+06	1.7	1.85	1	1.54e-01	1.42	1.50	94	2.47
4	24	2.6214e+05	1.0923e+04	3.7465e+06	1.7	1.85	1	1.58e-01	1.38	1.50	92	2.53
5	30	2.6214e+05	8.7381e+03	2.9972e+06	1.7	1.87	1	1.34e-01	1.63	1.87	87	2.15
6	36	2.6214e+05	7.2818e+03	2.4976e+06	1.7	1.84	1	1.16e-01	1.89	2.25	84	1.85
7	42	2.6214e+05	6.2415e+03	2.1408e+06	1.7	1.86	1	1.05e-01	2.08	2.62	79	1.68
8	48	2.6214e+05	5.4613e+03	1.8732e+06	1.7	1.85	1	9.37e-02	2.34	3.00	78	1.49
9	54	2.6214e+05	4.8545e+03	1.6651e+06	1.7	1.86	1	9.01e-02	2.43	3.37	72	1.44
11	60	2.6214e+05	4.3691e+03	1.4986e+06	1.7	1.84	1	8.28e-02	2.64	3.75	70	1.32
12	72	2.6214e+05	3.6409e+03	1.2488e+06	1.7	1.88	1	7.34e-02	2.99	4.50	66	1.17

Table 4: DOE ExaWind Application (ABL): NekRS/Nek5000 Strong Scaling Performance on Summit, Fugaku on a fixed domain $\Omega = [0,4]^3$. Restarting with initial condition at time= 6 hours, performed for 200 steps and the averaged timing per step, t_{step}, is measured in seconds for 101–200 steps with the timestep size Δt in nondimension and Δt^* in physical unit. Ω represents the computational domain in nondimension. The real time ratio is $r_t = t_{step}/\Delta t^*$ where $\Delta t^* = (L/u)\Delta t$ with the characteristic length, L=100m, and the stream velocity, 8m/s.

Nek5000 on Fugaku (Strong Scaling) ABL Flow										
node	core	E	N	n	n/core	t_{step}	R	R$_{\text{ideal}}$	P$_{\text{eff}}$	MPIIO mesh read (sec)
128	4096	2.6214e+05	7	89915392	21952	3.05e-01	1.00	1	100	3.8
256	8192	2.6214e+05	7	89915392	10976	2.13e-01	1.43	2	71	22
512	16384	2.6214e+05	7	89915392	5488	1.34e-01	2.27	4	56	0.12E+03
1024	32768	2.6214e+05	7	89915392	2744	9.48e-02	3.21	8	40	0.55E+03

Nek5000 on Fugaku (Weak Scaling) 17× 17 Rod Bundle										
node	core	E	N	n	n/core	t_{step}	R	R$_{\text{ideal}}$	P$_{\text{eff}}$	MPIIO mesh read (sec)
106	3392	277000	7	9.47340e+07	4.0e+05	1.462e-1	1	1	100	2.1
424	13568	1108000	7	3.78936e+08	4.0e+05	1.875e-1	0.77	1	77	5.0
1272	40704	3324000	7	1.13680e+09	4.0e+05	1.910e-1	0.76	1	76	0.56E+03
4664	149248	12188000	7	4.16829e+09	4.0e+05	-	-	-	-	not complete
11660	373120	30470000	7	1.04207e+10	4.0e+05	-	-	-	-	not complete
23320	740240	60940000	7	2.08414e+10	4.0e+05	-	-	-	-	not complete
46640	1492480	121880000	7	4.16829e+10	4.0e+05	-	-	-	-	not complete

Table 5: Nek5000 on Fugaku: Strong scaling for ABL flows; Weak scaling for 17×17 rod bundle simulation.

In fact, we observe the timing increase in MPIIO mesh reading at the medium scale for ABL and rod1717 cases as shown in Table 5. To address this issue, we have written a parallel reader that does not use MPIIO. With this reader, a user-defined number of MPI ranks (e.g., $P_r = 1000$) will open a file and index to the appropriate line in the file. These ranks are uniformly distributed over the full P ranks in the simulation. Each of these ranks reads its segment and we then invoke Nek5000's *gslib*-based crystal-router (all-to-all) to migrate the data from the readers to the consumers. The *gslib* crystal-router [16] code is well tested and has scaled to millions of ranks.

We observe that these cases exhibit excellent strong scaling to all of Summit, pointing to $n/P \approx 2.5$M as the 80% efficiency level for the V100s. Note that, save for the anomalous DNS 5×5 case, this value of n/P is consistent with our previous results. As discussed in [14, 13], the leading indicator of parallel scalability for a given algorithm-architecture coupling is n/P, rather than the number of processing units, P.

CASE (Strong Scaling): DOE ExaSMR Turbulent flow for 17× 17 Rod Bundle								
node	core per node	core	E	N	n	n/core	E/core	input mesh size
8.2944e+04	32	2.6542e+06	1.7562e+08	7	6.0237e+10	2.2695e+04	66	23GB
7.9488e+04	32	2.5436e+06	1.7562e+08	7	6.0237e+10	2.3682e+04	69	23GB
3.9744e+04	32	1.2718e+06	1.7562e+08	7	6.0237e+10	4.7363e+04	138	23GB
1.9872e+04	32	6.3590e+05	1.7562e+08	7	6.0237e+10	9.4727e+04	276	23GB
8.2944e+04	40	3.3178e+06	1.7562e+08	7	6.0237e+10	1.8156e+04	52	23GB
7.9488e+04	40	3.1795e+06	1.7562e+08	7	6.0237e+10	1.8945e+04	55	23GB
3.9744e+04	40	1.5898e+06	1.7562e+08	7	6.0237e+10	3.7891e+04	110	23GB
1.9872e+04	40	7.9488e+05	1.7562e+08	7	6.0237e+10	7.5781e+04	220	23GB
8.2944e+04	48	3.9813e+06	1.7562e+08	7	6.0237e+10	1.5130e+04	44	23GB
7.9488e+04	48	3.8154e+06	1.7562e+08	7	6.0237e+10	1.5788e+04	46	23GB
3.9744e+04	48	1.9077e+06	1.7562e+08	7	6.0237e+10	3.1576e+04	92	23GB
1.9872e+04	48	9.5386e+05	1.7562e+08	7	6.0237e+10	6.3151e+04	184	23GB

Table 6: Rod1717: Nek5000 Test Case Sizes on Fugaku.

CASE (Strong Scaling): DOE NEAMS Full-Core Pebble Bed Reactor for 350K pebbles								
node	core per node	core	E	N	n	n/core	E/core	input mesh size
8.2944e+04	32	2.6542e+06	9.8782e+07	7	3.3882e+10	1.2765e+04	37	78GB
7.9488e+04	32	2.5436e+06	9.8782e+07	7	3.3882e+10	1.3321e+04	38	78GB
3.9744e+04	32	1.2718e+06	9.8782e+07	7	3.3882e+10	2.6641e+04	77	78GB
1.9872e+04	32	6.3590e+05	9.8782e+07	7	3.3882e+10	5.3282e+04	155	78GB
8.2944e+04	40	3.3178e+06	9.8782e+07	7	3.3882e+10	1.0212e+04	29	78GB
7.9488e+04	40	3.1795e+06	9.8782e+07	7	3.3882e+10	1.0656e+04	31	78GB
3.9744e+04	40	1.5898e+06	9.8782e+07	7	3.3882e+10	2.1313e+04	62	78GB
1.9872e+04	40	7.9488e+05	9.8782e+07	7	3.3882e+10	4.2626e+04	124	78GB
8.2944e+04	48	3.9813e+06	9.8782e+07	7	3.3882e+10	8.5103e+03	24	78GB
7.9488e+04	48	3.8154e+06	9.8782e+07	7	3.3882e+10	8.8803e+03	25	78GB
3.9744e+04	48	1.9077e+06	9.8782e+07	7	3.3882e+10	1.7761e+04	51	78GB
1.9872e+04	48	9.5386e+05	9.8782e+07	7	3.3882e+10	3.5521e+04	103	78GB

Table 7: 350k pebbles: Nek5000 Test Case Sizes on Fugaku.

As shown in Table 5, we have made the following measurements:

- Strong scaling for ABL flow (E=64^3)
- Weak scaling for 17×17 rod bundle, with meshes using 0.2 million∼121 million elements (total 96 million∼41 billion grid points)

We had hoped to have more Fugaku timings for larger extension of the cases above but had difficulty with MPIIO mesh reading and were not able to get more simulations done other than Table 4–5.

Given the available medium-scale data, we can draw the following conclusions. From Table 4, it is clear that $n_{0.8} > 10,000$, but we cannot conclude that it is $< 20,000$ because we do not yet have data points for the cases with 40,000 points per rank. We see that Nek5000 with 8192 cores on Fugaku performs about the same as 17 V100s on Summit with (the highly-tuned) NekRS code. From Table 5 we see the mesh-read issues that we have had as we tried to scale to larger problems. We are hoping that our new parallel reader will fix this issue.

3.3 MFEM performance on Fugaku

In the beginning of 2021, the CEED team at LLNL ported the MFEM library to the Fugaku architecture and performed an initial performance evaluation. One of the main goals of this work was to get a preliminary impression of the level of performance achievable with the A64FX processor from Fujitsu, using pre-existing optimized code. In particular, we are interested in comparing the performance of the A64FX with NVIDIA's V100 GPU. On paper, the peak double-precision FLOPs of the V100 is approximately 2.5× that of the A64FX, and so we expect to see a significant benefit running compute-bound kernels on the V100. However, the two chips have similar memory bandwidth, and so we expect to see only a slight advantage for memory-bound kernels on the NVIDIA GPU.

Benchmark BP3. The BP3 benchmark which has been used is described in the CEED milestone report MS6 [12]. It is a scalar PCG with stiffness matrix: we solve $A\underline{u} = \underline{f}$, where A is the Poisson operator.

The code used as starting point consists of already vectorized CPU kernels: they represent our reference CPU and have been already improved and highly optimized, due to the bake-off runs and motivated by the deal.ii results, as explained in the CEED milestone report MS18 [8]. The CEED team added explicit vectorization over the elements to the MFEM code, in order to deliver a resulting two- to four-fold speed-up over the original results.

In order to optimize the microarchitecture-dependent bottlenecks that could be shared with the (weaker) streamed multi-processors of a GPU, the CEED team merged all the kernel development work carried out in the CEED milestone report MS8 [29] for this benchmark and ported the algorithm that have been demonstrated as successful there. It uses one third less of memory and few FLOP less per most inner `for`-loop bodies - which could make a difference. This way of proceeding allows a completely fair comparison, by using the exact same mathematical formulation and algorithms for the same benchmark BP3. The kernels have been ported to MFEM, through the `MFEM_FORALL` wrapper.

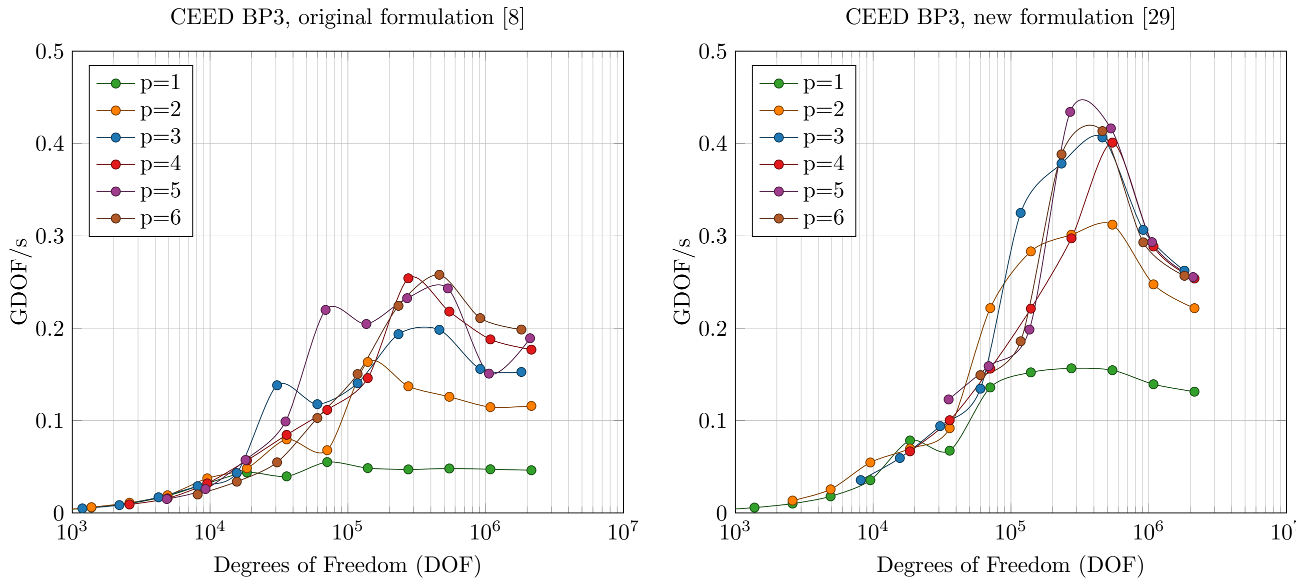


Figure 17: Baseline vectorized BP3 kernels, one Xeon 6130, 32 MPI ranks - gcc-10.2.0

Both the CPU and GPU code share now the same initial code before being mapped toward the GPU architecture that uses a block of inner threads inside the different scheduled blocks to divide the work of the inner `for`-loop bodies, whereas on the CPU side, these bodies are kept and forwarded to a single core, introducing explicit vectorization in order to take full advantage of its internal micro-architecture.

Figure 17 shows the benefit of the development: depending on the order and the number of degrees of freedom, a noticeable and non negligible speedup has been obtained by using tuned code for GPU and reusing it with explicit vectorization for an x86 CPU chip.

We have good confidence that for an MPI-based only programming model this approach delivers the best from each single core. However, a hybrid MPI+OpenMP model could be sought, in order to reduce the total number of MPI ranks, the memory footprint on the processor and take advantage of the cache hierarchy of the four core memory groups of the A64FX, which are connected with a NUMA ring network on chip. These developments require more changes than the current scope of our porting and benchmarking efforts.

A64FX SVE Vectorization. Achieving high performance on Fugaku requires explicit vectorization for the x86 architecture. Some amount of auto-vectorization is possible, but forcing the compiler to use vectorized instructions leads to the same 2× speedup we have usually seen on the other architectures: SSE, AVX, QPX, and VSX. Auto-vectorization on a SIMD length of 8 doubles can generally reach a 4× speedup compared to no vectorization at all.

Different compiler configurations were tested: Fujitsu's C++ compiler (FCC 4.4.0a 20211027), as well as the GNU GCC 10.2.0. Figure 18 shows the difference in performances in terms of giga-degrees-of-freedom per second (GDOF/s) between the two compilers. Since the GNU GCC compiler delivered significantly faster performance for this test case, all subsequent results have been run using GCC.

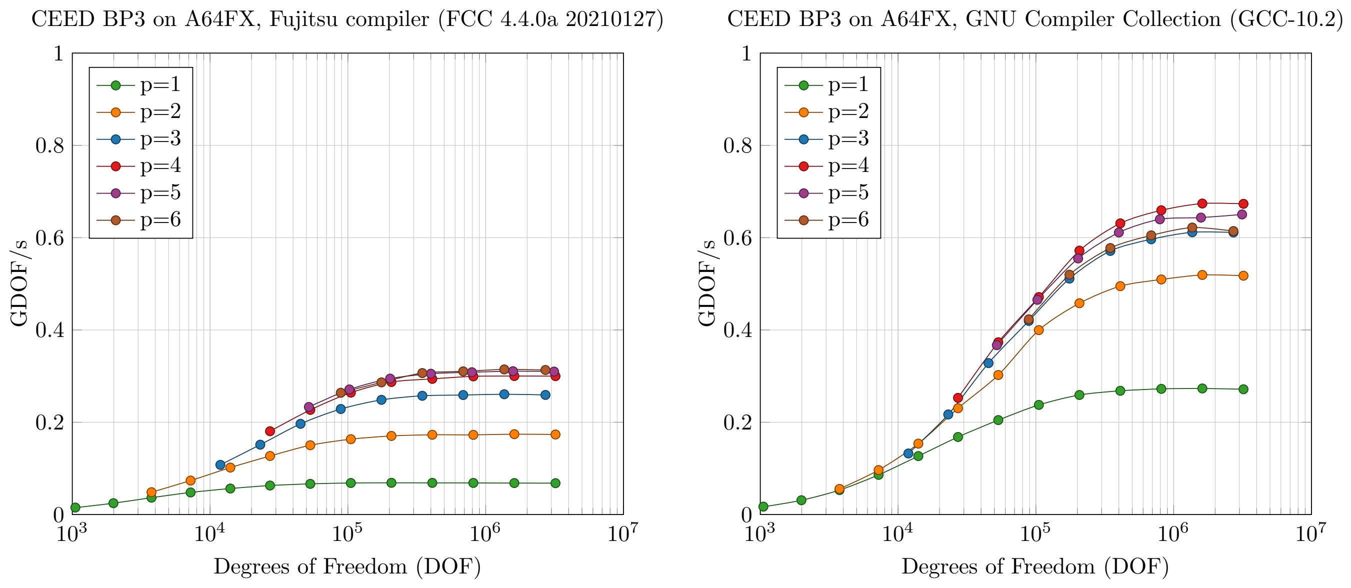


Figure 18: CEED BP3 results on A64FX with Fujitsu's FCC compiler (4.4.0a) and GNU Compiler Collection (10.2.0)

Benchmark Results. We tested 3 architectures: 1) Intel Xeon Gold 6130 2.1GHz with 16 cores, 2) A64FX with 48 cores 2.0GHz and 3) NVIDIA Quadro GV100. Figure 19 shows the result for BP3, orders 1 and 4 for all three architectures on the same plot. For low orders, the V100 and A64FX chips are similar, but for high orders the V100 is 2-3× faster.

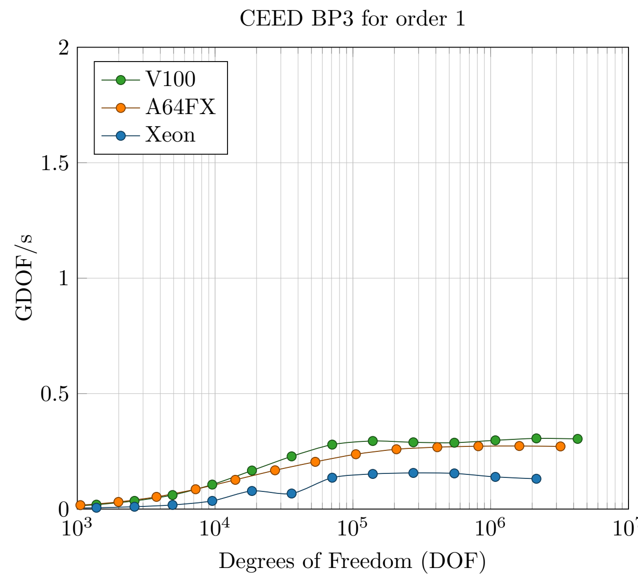
Figure 20 shows on the left all orders for one A64FX, compared to one V100 on the right, where a significantly bigger boost from high-order methods can be observed.

Many optimizations still remain to be performed, and the code could be further modified to extract the highest possible performance from this architecture. The results are currently only based on one-core tuning, with no specific code dedicated to data locality or hidden latency, and no specific compiler optimizations.

4. CEED-4.0 RELEASE

The CEED distribution is a collection of software packages that can be integrated together to enable efficient discretizations in a variety of high-order applications on unstructured grids. CEED is using the Spack package manager for compatible building and installation of these software components, and also provides Docker images for containerized deployment and continuous integration.

Version CEED-4.0 released on March 31st, 2021 contains 12 integrated packages ranging from low-level modular libraries to applications, provided together within the CEED meta-package. We list these packages below, listing some highlights for each of them since the last release. Note that many of these packages have had more than one feature release since CEED-3.0, some of which were reported in [22]; the list below reflects only the latest releases since CEED-3.0.



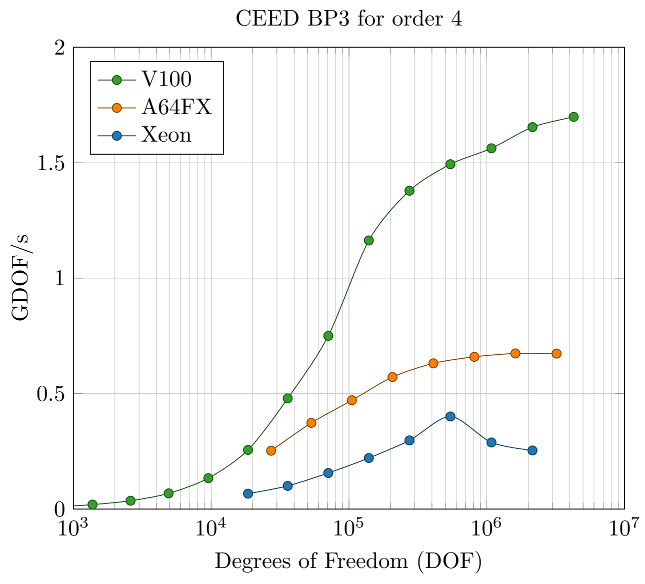
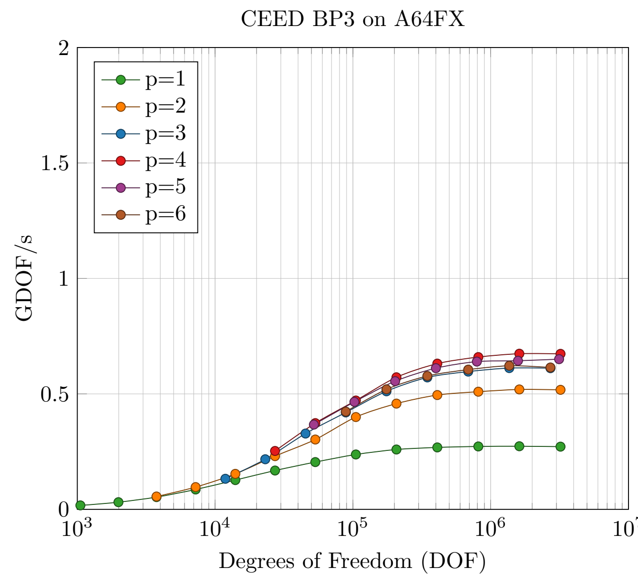


Figure 19: Orders 1 and 4 on V100, A64FX and Xeon architectures.



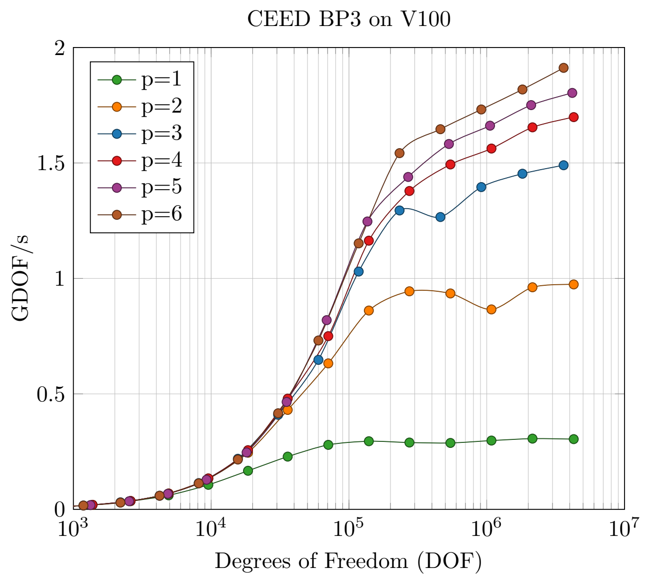


Figure 20: BP3 on Fujitsu's A64FX and NVIDIA's V100 architectures.

libCEED-0.8 $\mapsto$ Support for matrix assembly (mainly intended for low order and coarse grids), new HIP and MAGMA backends with kernel fusion, Julia and Rust interfaces, and static linking.

MAGMA-2.5.4 $\mapsto$ Support for CUDA 11 and Ampere GPUs, added BLAS kernels and optimizations needed for hipMAGMA port of MAGMA, bug fixes and performance improvements.

MFEM-4.2 $\mapsto$ Element and full assembly on GPUs, vectorization, HIP and CUDA improvements, new mesh optimization and discretization algorithms, AmgX, CVODES, CPardiso, SLEPc and ADIOS2 support and 18 new examples + miniapps.

NekRS-21.0 $\mapsto$ NekRS is a new C++ port of Nek5000 focused in GPUs. Version 21.0 adds support for Chebyshev-accelerated ASM and RAS smoothers, improved gather-scatter performance, stress formulation, initial guess projection, and ALE formulation to support moving meshes.

OCCA-1.1.0 ↦ Support for new ROCm releases, more flexible management of CUDA streams, and portability improvements.

PETSc-3.15 ↦ Support for Kokkos and Kokkos-Kernels on NVIDIA and AMD GPUs, consolidation of communication via the `PetscSF` layer with improved collective patterns and nonblocking pack/unpack via NVSHMEM backend (see [41]), new pipelined Krylov methods, and improved support for high order elements in the `DMPlex` mesh.

PUMI-2.2.5 ↦ Improved support for high-order nodal fields and solution transfer, extended the python interface, and bug fixes. See the release notes for details.

The CEED-4.0 distribution also contains stable packages that have not seen a new release since CEED-3.0, namely Laghos-3.0, Nek5000-19.0, Nekbone-17.0, NekCEM-c8db04b, and Remhos-1.0.

The distribution is tested on several platforms including Linux (RHEL and Ubuntu), Mac OSX, and the representative heterogeneous HPC machines OLCF Summit, LLNL Lassen, and LLNL Corona. See `https://ceed.exascaleproject.org/ceed-4.0/` for details about native installation using Spack and containerized use via Docker, Singularity, and Shifter.

5. LIBCEED-0.8 RELEASE

We have continued the development of libCEED, and as its interface has become more stable, our work has become more independent between backend development and tuning versus high-level integration. Our high-level integration has included language bindings that are now distributed with libCEED, including new Julia and Rust interfaces, along with integration of libCEED in higher-level libraries like MFEM and PETSc. Notable new features in the latest release, libCEED-0.8, include:

- The new `CeedOperatorLinearAssemble` facilitates assembly of sparse matrices on CPUs and GPUs, as is often desirable for low-order operators and coarse grids.
- Julia and Rust interfaces added, providing a nearly 1-1 correspondence with the C interface, plus some convenience features.
- New HIP MAGMA backends for hipMAGMA library users: `/gpu/hip/magma` and `/gpu/hip/magma/det`.
- New HIP backends for improved tensor basis performance: `/gpu/hip/shared` and `/gpu/hip/gen`.
- Static libraries can be built with `make STATIC=1` and the pkg-config file is installed accordingly.
- Examples: the solid mechanics example has updated with traction and non-affine boundary conditions as well as compact storage representations in the current configuration, similar to [11].

5.1 libCEED integration in Julia

The latest release of the libCEED library includes interfaces making the library accessible from the Julia and Rust languages. The Julia interface makes all of the features of the libCEED library accessible from Julia applications. Consequently, libCEED can be used from interactive REPL-based and notebook-based programming environments. Julia's metaprogramming and just-in-time compilation facilities are leveraged in order to allow the user to define single-source QFunctions written in Julia that can be used on both the CPU and GPU. Currently, user QFunctions written in Julia are compiled using `CUDA.jl` to run on NVIDIA GPUs. These features allow the user to use high-level linear algebra functionality, without incurring performance penalties. For example, the following code snippet can be used to define the "assemble diffusion" QFunction, which computes the quadrature data required to apply the diffusion operator:

```julia
@interior_qf build_qfunc = (
    ceed,
    dim=dim,
    (J, :in, EVAL_GRAD, dim, dim),
    (w, :in, EVAL_WEIGHT),
    (qdata, :out, EVAL_NONE, dim*(dim + 1)÷2),
    begin
        Jinv = inv(J)
        qdata .= setvoigt(w*det(J)*Jinv*Jinv')
    end,
)
```

In this example, the linear algebra functions `det` and `inv` compile to efficient dimension-specific code using only statically sized arrays.

5.2 libCEED integration in Rust

Rust is a new statically-typed programming language with precise memory semantics (no garbage collection), strong safety and concurrency features, and powerful static analysis. It has become a popular alternative to C and C++ in many performance-sensitive domains and we see it as a promising language for improving reliability in scientific computing. The new interface is relatively idiomatic, and includes a test suite, examples, and a documentation site. An extremely compelling feature of Rust is Cargo, the package manager, which provides ergonomic and reliable dependency management, including the ability to compile libCEED from source, thus significantly lowering the complexity of distributing and using libraries.

5.3 libCEED integration in MFEM

The integration of libCEED in MFEM brings many features into MFEM. A high-level interface in MFEM allows the user to easily describe partially assembled libCEED operators, and fully matrix-free libCEED operators directly using MFEM objects. The user only has to write the QFunctions, and all the libCEED function calls are transparently handled by the high level interface. Through this interface several `mfem::Integrator`s are already supported:

- `mfem::MassIntegrator` and `mfem::VectorMassIntegrator`: $\int(qu, v)$,
- `mfem::ConvectionIntegrator`: $\int(q \cdot \nabla u, v)$,
- `mfem::VectorConvectionNLFIntegrator`: $\int(q(\nabla u)u, v)$,
- `mfem::DiffusionIntegrator` and `mfem::VectorDiffusionIntegrator`: $\int(q\nabla u, \nabla v)$.

The support for these `mfem::Integrator`s allow to use libCEED transparently in several MFEM examples just by configuring the `mfem::Device`. The targeted architecture for libCEED can easily be configured by providing the string `ceed-cpu`, `ceed-cuda`, or `ceed-hip` to the `mfem::Device` for CPUs, NVIDIA GPUs, and AMD GPUs respectively. Specific libCEED backends can be selected using the colon symbol `:`, e.g. `ceed-cuda:/gpu/cuda/shared` to run with the `cuda-shared` backend of libCEED without having to change any code.

More recently an algebraic multigrid solver based on the libCEED integration has also been introduced in MFEM (`mfem::ceed::AlgebraicSolver`). Usable with any MFEM operator that supports libCEED, this preconditioner dramatically improves the performance over the Jacobi smoother (`mfem::OperatorJacobiSmoother`), see Table 8.

In order to provide libCEED support for an `mfem::Integrator`, it is sufficient to provide the following `struct` containing the data about the QFunctions and the evaluation mode of the trial and test functions:

```
/** This structure is a template interface for the Assemble methods of
    PAIntegrator and MFIntegrator. See ceed/mass.cpp for an example. */
struct OperatorInfo
{
   /** The path to the qFunction header. */
   const char *header;
   /** The name of the qFunction to build a partially assembled CeedOperator
       with a constant Coefficient. */
   const char *build_func_const;
   /** The qFunction to build a partially assembled CeedOperator with a constant
       Coefficient. */
   CeedQFunctionUser build_qf_const;
   /** The name of the qFunction to build a partially assembled CeedOperator
```

order	Jacobi	Algebraic	#dofs
1	49	22	31841
2	127	18	241857
3	270	20	802081
4	414	18	1884545
5	564	18	3661281
6	715	20	6304321
7	873	20	9985697
8	1031	18	14877441

Table 8: Number of iterations for the Jacobi and the algebraic matrix-free solvers in MFEM using libCEED on a 3D mesh (`fichera.mesh`) solving CEED BP3.

```
       with a variable Coefficient. */
   const char *build_func_quad;
   /** The qFunction to build a partially assembled CeedOperator with a variable
       Coefficient. */
   CeedQFunctionUser build_qf_quad;
   /** The name of the qFunction to apply a partially assembled CeedOperator. */
   const char *apply_func;
   /** The qFunction to apply a partially assembled CeedOperator. */
   CeedQFunctionUser apply_qf;
   /** The name of the qFunction to apply a matrix-free CeedOperator with a
       constant Coefficient. */
   const char *apply_func_mf_const;
   /** The qFunction to apply a matrix-free CeedOperator with a constant
       Coefficient. */
   CeedQFunctionUser apply_qf_mf_const;
   /** The name of the qFunction to apply a matrix-free CeedOperator with a
       variable Coefficient. */
   const char *apply_func_mf_quad;
   /** The qFunction to apply a matrix-free CeedOperator with a variable
       Coefficient. */
   CeedQFunctionUser apply_qf_mf_quad;
   /** The EvalMode on the trial basis functions. */
   EvalMode trial_op;
   /** The EvalMode on the test basis functions. */
   EvalMode test_op;
   /** The size of the data at each quadrature point. */
   int qdatasize;
};
```

For example, the specialization of the `struct mfem::ceed::OperatorInfo` for `mfem::VectorConvectionNLFIntegrator`:

```
struct NLConvectionOperatorInfo : public OperatorInfo
{
   NLConvectionContext ctx;
   NLConvectionOperatorInfo(int dim)
   {
      header = "/nlconvection_qf.h";
      build_func_const = ":f_build_conv_const";
      build_qf_const = &f_build_conv_const;
      build_func_quad = ":f_build_conv_quad";
      build_qf_quad = &f_build_conv_quad;
      apply_func = ":f_apply_conv";
      apply_qf = &f_apply_conv;
      apply_func_mf_const = ":f_apply_conv_mf_const";
      apply_qf_mf_const = &f_apply_conv_mf_const;
      apply_func_mf_quad = ":f_apply_conv_mf_quad";
      apply_qf_mf_quad = &f_apply_conv_mf_quad;
      trial_op = EvalMode::InterpAndGrad;
      test_op = EvalMode::Interp;
      qdatasize = dim * dim;
   }
};
```

This data is then utilized to either create a partially assembled libCEED operator, i.e. `mfem::ceed::PAIntegrator`, or fully matrix-free libCEED operator, i.e. `mfem::ceed::MFIntegrator`. Through inheriting from either of these classes, a class can use the inherited `Assemble` method to automatically generate an MFEM operator using transparently libCEED; requiring from the user to only write code for the libCEED QFunctions. The following code shows the full implementation of a partially assembled `mfem::VectorConvectionNLFIntegrator` operator using libCEED, the high level interface allows to keep the amount of code minimal:

```
PAVectorConvectionNLFIntegrator::PAVectorConvectionNLFIntegrator(
   const mfem::FiniteElementSpace &fes,
   const mfem::IntegrationRule &irm,
   mfem::Coefficient *Q)
   // Indicates that we want to build a partially assembled operator.
   : PAIntegrator()
{
   // Initialize the information to build the libCEED operator.
   NLConvectionOperatorInfo info(fes.GetMesh()->Dimension());
   // A single call to Assemble inherited from PAIntegrator takes care of all the rest.
   Assemble(info, fes, irm, Q);
}
```

Any `mfem::Integrator` that supports libCEED comes with the full set of libCEED features readily available. This includes support for arbitrary polynomial order, basis functions, quadrature rules, and simplex and hex meshes in 1D, 2D, 3D, see Figure 21 and Figure 22 for a comparison of the performance of tets and hexes on an Intel Xeon E5-2695 CPU, and Figure 23 for performance on a NVIDIA V100.

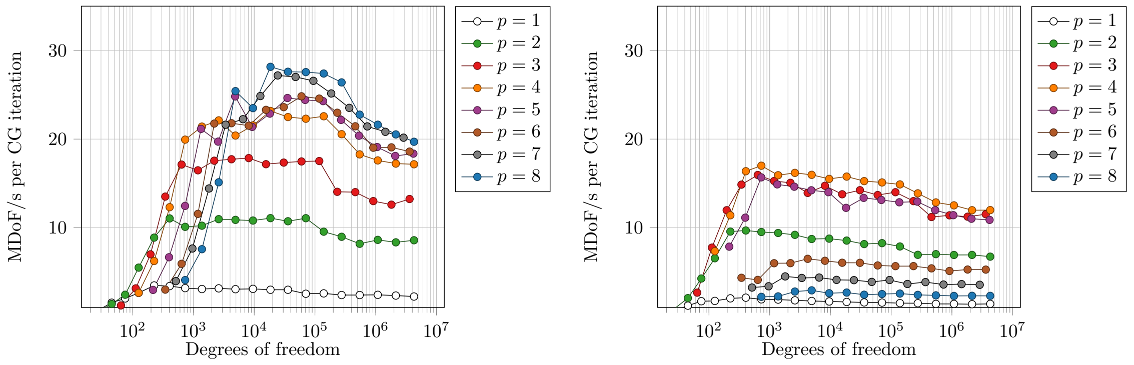
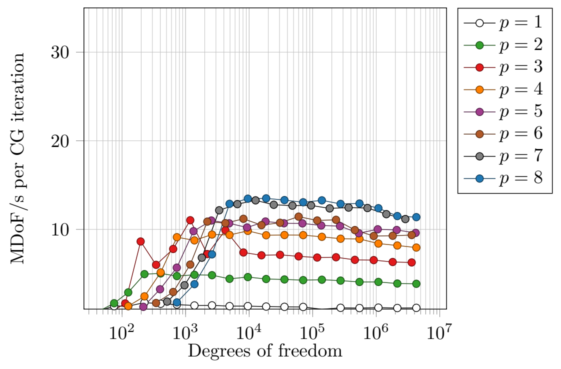


Figure 21: Comparison of the performance of partially assembled operators to solve CEED BP3 on hexes (left) and tets (right) using an Intel Xeon E5-2695 CPU.

6. APPLICATION COLLABORATIONS

6.1 Breakthrough: NEAMS all-hex meshing for 350K pebbles

We developed an all-hex meshing strategy for the interstitial space in beds of densely-packed spheres that is tailored to turbulent flow simulations based on spectral element method (SEM). The SEM achieves resolution though elevated polynomial order, N, and requires two to three orders of magnitude fewer elements than standard FEM approaches, which places stringent requirements on mesh quality and conformity. Our meshing algorithm is based on a Voronoi decomposition of the sphere centers. Facets of the Voronoi cells are tessellated into quads that are swept to the sphere surface to generate a high-quality base mesh. Refinements to the algorithm include edge-collapse to remove slivers, node insertion to balance resolution, localized refinement in



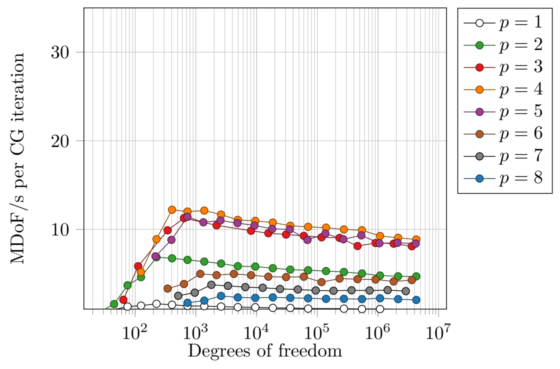
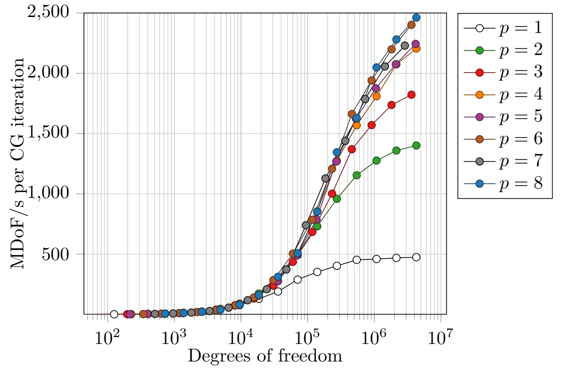


Figure 22: Comparison of the performance of fully matrix-free operators to solve CEED BP3 on hexes (left) and tets (right) using an Intel Xeon E5-2695 CPU.



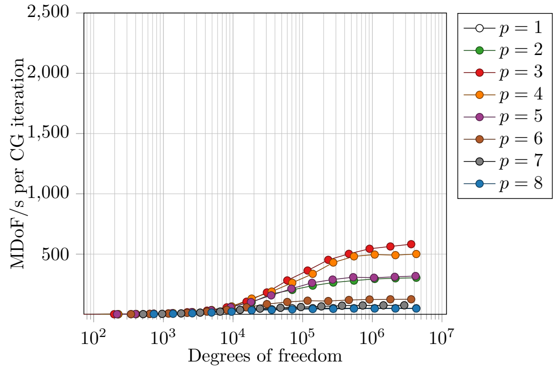


Figure 23: Comparison of the performance of partially assembled operators to solve CEED BP3 on hexes (left) and tets (right) using a NVIDIA V100 GPU.

the radial direction about each sphere, and mesh optimization. We demonstrate geometries with 10^2–10^5 spheres using ≈ 300 elements (for three radial layers) per sphere, along with mesh quality measurements, flow simulations, validation, and performance.

We have applied our all-hex meshing algorithm to several configurations listed in Table 9. We note that the number of elements per sphere for the three-layer-per-sphere approach leads to about 300 elements per

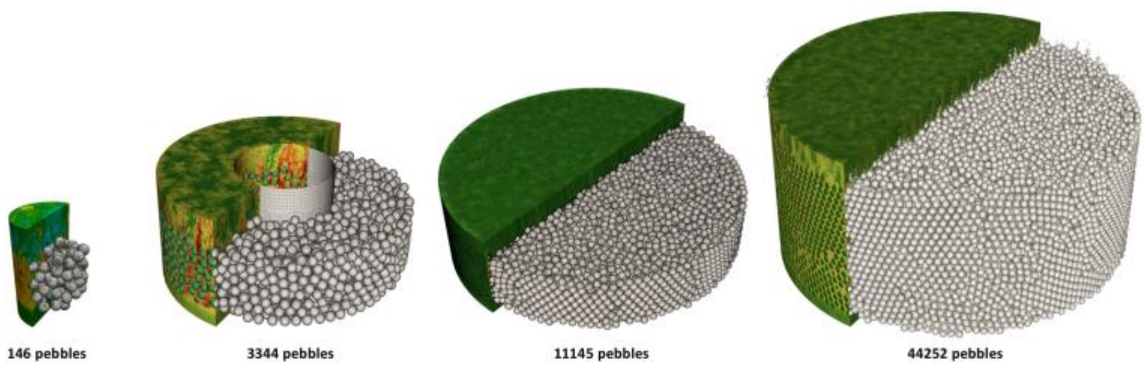


Figure 24: Pebble meshes and simulations.

Statistics for Several Multi-Sphere Configurations					
Case	$\mathcal{N}$	Source	Container	E	$E/\mathcal{N}$
cyl-146	146	Experiment	cylinder	62,132	425.56
box-1053	1,053	StarCCM+	box	376,828	250.72
cyl-1568	1,568	StarCCM+	cylinder	524,386	334.43
cyl-3260	3,260	StarCCM+	cylinder	1,121,214	343.93
ann-3344	3,344	Project Chrono	annulus	1,133,446	338.95
cyl-11k	11,145	Project Chrono	cylinder	3,575,076	320.78
cyl-44k	44,257	Project Chrono	cylinder	13,032,440	294.47
cyl-49k	49,967	Project Chrono	cylinder	14,864,766	297.49
ann-127k	127,602	StarCCM+	annulus	39,249,190	307.52
ann-350k	352,625	StarCCM+	annulus	98,782,067	280.13

Table 9: List of cases meshed to date including the number of spheres, $\mathcal{N}$, the data source, domain shape, number of elements, E, and number of elements per sphere.

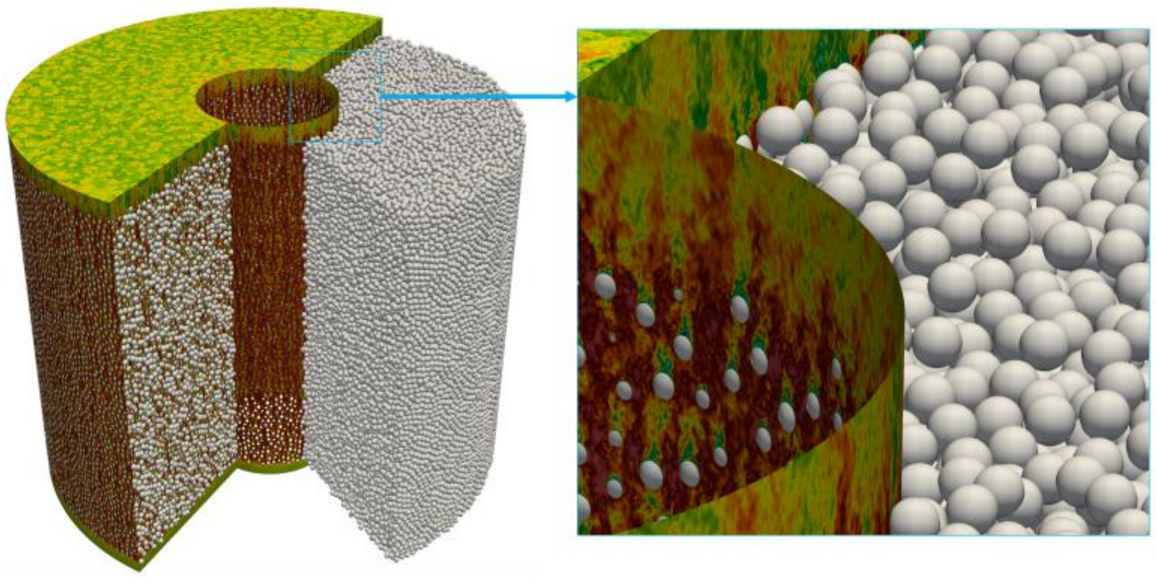


Figure 25: Turbulent flow in an annular packed bed with $\mathcal{N} = 352625$ spheres meshed with $E = 98,782,067$ spectral elements of order $N = 7$ ($n = 34.7$ billion gridpoints). This pebble simulation has performed up to 4608 nodes using 27648 V100s on Summit. The average pressure iterations per step is currently 13.7 when using CHEBY+ASM with 2 Chebyshev smoothing steps.

sphere, including the elements in the inlet- and exit-flow regions. Figure 24 shows several of the corresponding sphere configurations along with axial-flow velocity distributions. Regions of order and disorder are evident in the positions of the spheres along the domain wall for the case of $\mathcal{N} = 44252$. Such ordered packing has a direct influence on the flow conditions near the domain boundary and is an important consideration for thermal-hydraulics analysis. All of the cases shown here were run with NekRS using the NVIDIA V100s on the OLCF Summit supercomputer at Oak Ridge National Laboratory.

Figure 25 shows a close-up of the results for the case of $\mathcal{N} = 352625$ using $E = 99$ million elements of order $N = 7$ (n=34.7 billion gridpoints). This pebble simulation has performed up to 4608 nodes using 27648 V100s on Summit. We are currently performing different number of Chebyshev smoothing steps. The average pressure iterations per step is currently 13.7 when using CHEBY+ASM with 2 Chebyshev smoothing steps.

As seen in Figure 26, left, the simulation using CHEBY+ASM with 2 Chebyshev smoothing steps takes

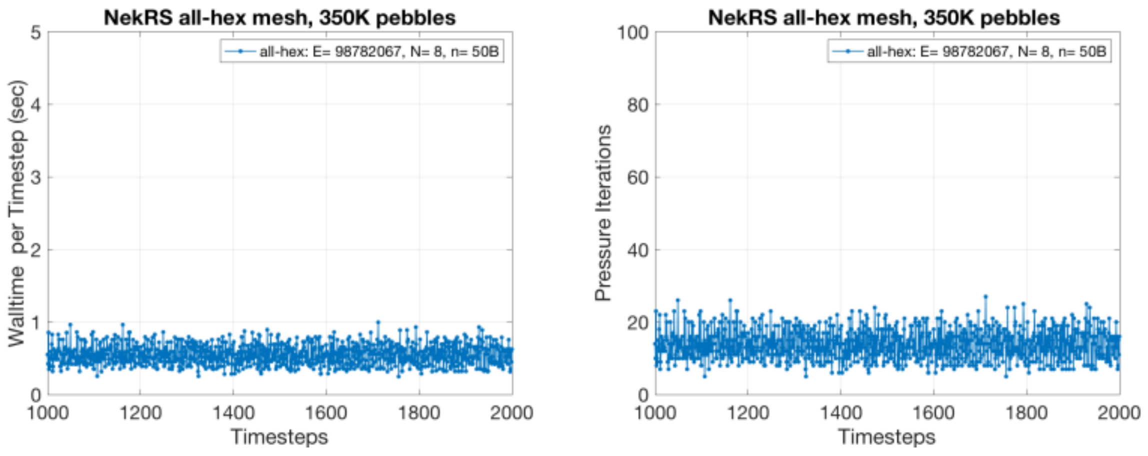


Figure 26: NekRS simulation using 3840 nodes (23040 V100s) on Summit, demonstrating the walltime per timestep and pressure iterations for 352625 pebbles. The timestep size is Δt=4E-04. Characteristic-based BDF2 with 1 substeps is used for timestepping and Chebyshev-accelerated spectral element multigrid preconditioner with *hypre* AMG for coarse grid solve is used for pressure solve. Tolerances for pressure and velocity are 1e-4 and 1e-6, respectively. The averaged CFL is 0.54 for 1001-2000 steps.

NekRS performance using all-hex mesh for 352625 pebbles											
mesh	node	gpu	E	E/gpu	N	n	n/gpu	v_i	p_i	CFL	t_{step}
all-hex	3840	23040	9.8721e+07	1.0289e+04	8	5.057e+10	2.19e+06	3.6	13.7	1.73	0.54

Table 10: NekRS simulations on Summit using 3840 nodes (23040 GPUs) for 352625 pebbles with $Re = 500$, performed for 2000 steps and the averaged timing per step, t_{step}, is measured in seconds for 1001–2000 steps. The timestep size is Δt=4.E-04. v_i and p_i represent the averaged values of the velocity components and pressure. Chebyshev-accelerated spectral element multigrid preconditioner with *hypre* AMG for coarse grid solve is used for pressure solve. Tolerances for pressure and velocity are 1e-4 and 1e-6, respectively. The averaged CFL is 0.54 for 1001-2000 steps.

about 0.54 second per step when running on 3840 nodes (23040 V100s), which corresponds to 2.1 million points per V100. The case using the full system (4608 nodes) is in job queue on Summit at the moment.

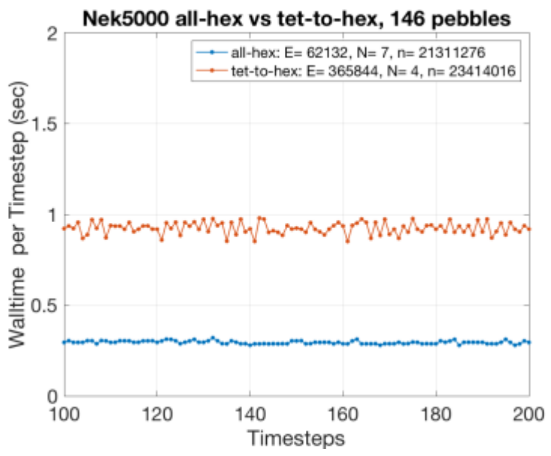
In Figure 26, right, we see that the average number of pressure iterations is around 13.7 per step, which we expect to be able to reduce further. Part of the issue stems from the highly compressed elements that are squeezed between the sphere (nominal) contact points. As discussed in the next section, a future development for our code will be to replace the flattened spheres at the contact points with a chamfered bridge of solid material that will join the spheres (again, avoid the singularity of a point contact). Preliminary experience suggests that this leads to a significant reduction in the condition number of the discrete pressure-Poisson operator and should improve runtimes.

The overall performance statistics for the 350K case are summarized in Table 10. A similar set of statistics for the $\mathcal{N} = 146$ case is provided in Table 11 for both the all-hex meshing strategy and the tet-to-hex approach [40]. At comparable resolution, n, the all-hex approach has a lower CFL (for the same timestep size) and lower wall-clock time per timestep because of the lower pressure iteration count.

NekRS Coupling with OpenMC and MOOSE under Cardinal Framework. In additional to fluid only simulations, coupled simulations have been performed with Cardinal [27], a multi-physics platform developed in the NEAMS program. Cardinal couples NekRS to OpenMC, a Monte Carlo solver, and MOOSE, a multi-physics framework that has been used to solve the complex physics involved in fuel performance:

Nek5000 performance comparison between hex and tet-to-hex meshes for 146 pebbles												
mesh	node	core	E	E/core	N	n	n/core	v_i	p_i	CFL	t_{step}	R
all-hex	16	672	62132	92	7	21311276	3.1713e+04	2.0	5	0.92	0.2954	1
tet-to-hex	16	672	365844	544	4	23414016	3.4842e+04	1.1	17	2.11	0.9251	3.13

Table 11: Nek5000 performance on Summit for 146 pebbles simulations with $Re = 5000$, performed for 200 steps and the averaged timing per step, t_{step}, is measured in seconds for 101–200 steps with the timestep size Δt=8.00E-04. v_i and p_i represent the averaged values of the velocity components and pressure. R represents the ratio of hex to tet-to-hex for t_{step}. Characteristic-based BDF2 with 2 substeps is used for timestepping and overlapping Schwarz smoothing with spectral element multigrid preconditioning with *hypre* AMG for coarse grid solve is used for pressure solve. Tolerances for pressure and velocity are 1e-4 and 1e-6, respectively.



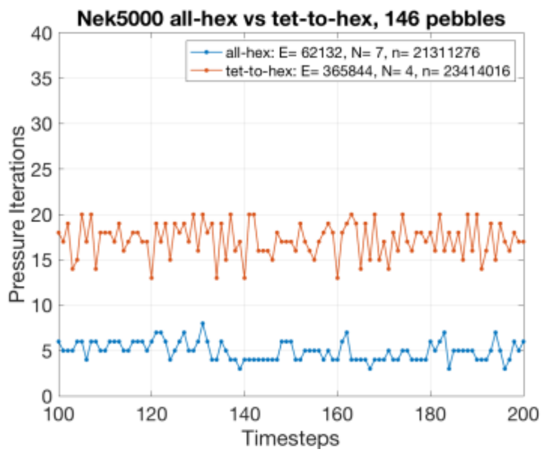


Figure 27: Nek5000 simulation walltime per timestep and pressure iterations for 146 pebbles with the timestep size Δt=8.00E-04. Characteristic-based BDF2 with 2 substeps is used for timestepping and overlapping Schwarz smoothing with spectral element multigrid preconditioning with *hypre* AMG for coarse grid solve is used for pressure solve. Tolerances for pressure and velocity are 1e-4 and 1e-6, respectively.

1. NekRS simulates the flow and temperature distribution around the pebbles on the GPUs.
2. OpenMC solves neutron transport on the CPU using 6 threads for each MPI rank.
3. MOOSE solves the unsteady conduction in the fuel pebbles on the CPU, and can incorporate more complex fuel performance models (BISON) including fission production and transport.

The Cardinal team has performed on Summit with 6 MPI ranks per node for OpenMCS and MOOSE and 6 GPUS per node for NekRS. The coupling simulations have been performed for a range of pebble sizes:

- 11,000 pebbles on 100 nodes.
- 127,000 pebbles on 1000 nodes.
- 350,000 pebbles on 3000 nodes.

In all cases the solution transfer between the physics did not consume more than 5.3% of runtime and the NekRS solve consumed at least 80% of the runtime (timings based on 500 NekRS steps).

6.2 Breakthrough: ExaSMR simulations using more than 600 million elements

We improved the PARRSB mesh partitioning algorithm to support more than 180M elements (92B grid points). Nek5000/NekRS currently support simulations at these scales. Our developments include 64-bit edge indexing and tools for scalable mesh extrusion and connectivity generation.

We are currently capable of supporting meshes over 600 million elements for Nek5000/RS. Our tests have been using a relatively small polynomial order with $N = 4$ to keep the problem size small for being able to debug fast at a reduced resource usage and minimal queue time on the capability test. We tested a full-core reactor mesh, consisting of 37 of 17×17 rods, having more than 614 million elements ($E = 614,940,000$) with $N = 4$, total 39 billion grid points. We used 3280 nodes (137760 MPI ranks), having n/P=285,686 grid points per rank, which corresponds to 1.9 million grid points per GPU for NekRS run.

6.3 ExaWind bake-off performance

We investigated accurate modeling and efficient simulations of atmospheric boundary layer flows. We present LES turbulence models for stable convection flows for benchmark problems, provided with validation and convergence studies. We are currently considering three different codes, Nek5000/RS, NaluWind and AMR-Wind, representing unstructured high-order, unstructured low-order and structured low-order discretizations and conducting their performance on various CPU and GPU platforms including Eagle, Summit and Fugaku. In this section, we focus on demonstrating NekRS performance on Summit.

In collaboration with ExaWind team, we have been performing the cross-verification and validation of our LES results and corresponding wall models. We perform a number of scaling studies to compare the performance of several ABL codes on CPU and GPU platforms. A well-documented stably stratified atmospheric boundary layer benchmark problem that can be used for these purposes, namely, for model and code intercomparison, is the Global Energy and Water Cycle Experiment (GEWEX) Atmospheric Boundary Layer Study (GABLS).

We initiated with convergence studies. Table 12 demonstrates varying resolutions representing indicating the minimum grid spacings in vertical and horizontal direction. We used the following parameters:

- Domain size 400×400×400m
- Geostrophic wind speed in the x-direction of 8m/s
- Reference potential temperature of 263.5K
- Resolution: 640 elements (8×8 in the two horizontal directions and 10 in the vertical direction). We used a polynomial order of 7 in each direction, corresponding to an effective resolution of 64^3. An additional simulation with polynomial order of 11 was also performed and both results are reported.
- Initial conditions: constant x-velocity equal to geostrophic wind speed, potential temperature at the surface equal to $265K$ up to $100m$ and after that linearly increasing with rate $0.01K/m$)
- Boundary conditions: Both no-slip as well as wall functions based on log-law were tried. Better agreement with benchmark was obtained using no-slip boundary condition at lower wall - with no need for additional resolution - and for this reason only these results are shown here.
- LES model: In these preliminary simulations we have used a relaxation LES model based on a high-pass filter (Schlatter et al.) which implemented through a relaxation term in the momentum equation. The use of more sophisticated physics-based LES models can be a topic of future work.

In this report, we focus on demonstrating NekRS performance on Summit GPUs. Table 13 shows NekRS strong scaling performance for $E = 64^3$, $E = 128^3$, and $E = 256^3$ with $N = 8$, which correspond to AMRWind grid resolutions of 512^3, 1024^3, and 2048^3, respectively. Table 13 demonstrates NekRS consistently achieves approximately 80% parallel efficiency using ∼2.2 million grid points per GPU. With ExaWind team, the real time ratio r_t is used as metric to compare the performance between the codes. The r_t increases, as expected, as the resolution increases and the average-time per step t_{step} reasonably scales as the number of GPUs increases for each resolution.

Convergence Study (non-uniform in vertical)		
E	horizontal	vertical
8x8x10	6.25m	0.75m–6.25m
32x32x10	1.56m	0.75m–6.25m

Strong Scaling starting from below (fixed domain, increasing resolution)					
gpu	E	E/gpu	n/gpu ($N = 7$)	horizontal	vertical
1	16x16x32	8192	2.8M	3.125m	1.56m
2	32x32x32	16384	5.6M	1.56m	1.56m
≥16	64x64x64	16384	5.6M	0.78m	0.78m
≥128	128x128x128	16384	5.6M	0.39m	0.39m
≥1024	256x256x256	16384	5.6M	0.19m	0.19m

Weak Scaling (with increasing domain, keeping same mesh density)				
gpu	E	E/gpu	n/gpu ($N = 7$)	horizontal, vertical
16	64x64x64	16384	5.6M	0.78m
64	128x128x64	16384	5.6M	0.78m
256	256x256x64	16384	5.6M	0.78m
1024	512x512x64	16384	5.6M	0.78m
gpu	E	E/gpu	n/gpu ($N = 8$)	horizontal, vertical
16	64x64x64	16384	8.3M	0.78m
64	128x128x64	16384	8.3M	0.78m
256	256x256x64	16384	8.3M	0.78m
1024	512x512x64	16384	8.3M	0.78m

Weak Scaling (with increasing domain, keeping same mesh density)				
gpu	E	E/gpu	n/gpu ($N = 8$)	horizontal, vertical
30	64x64x64	8738	4.4M	0.78m
120	128x128x64	8738	4.4M	0.78m
480	256x256x64	8738	4.4M	0.78m
1920	512x512x64	8738	4.4M	0.78m
gpu	E	E/gpu	n/gpu ($N = 8$)	horizontal, vertical
60	64x64x64	4369	2.2M	0.78m
240	128x128x64	4369	2.2M	0.78m
960	256x256x64	4369	2.2M	0.78m
3840	512x512x64	4369	2.2M	0.78m

Table 12: Convergence study with multiple resolution with nonuniform element size in vertical direction. Total physical simulation time is 9 hrs; **Strong Scaling**: We decrease E/gpu by 16384, 14000, 12000, 10000, 8000, 6000. Initial condition at t=6hrs and run 200 timesteps, and get the averaged time per step using 101-200 steps. **Weak Scaling** with increasing domain size, keeping the same mesh density.

6.4 Adaptive simulation workflow for RF-SciDAC

As indicated in Figure 28, tokamak fusion systems are geometric complex systems made-up of a large number of components. Working jointly with the SciDAC Center for Integrated Simulation of Fusion Relevant RF Actuators [7], CEED has been developing an MFEM-based simulation workflow that can effectively perform adaptive simulations of RF antenna arrays in fusion tokamak systems. The key steps in this simulation workflow include:

- Accessing the component geometric descriptions.
- Building the analysis model geometry from the component geometry descriptions.
- Associating the required analysis attributes (material properties, loads, boundary conditions and initial conditions) with that analysis geometry.
- Generating an initial mesh of the analysis geometry and preparing the input information needed by MFEM.
- Performing the MFEM analyses on that mesh and applying adaptive mesh control to improve the mesh as needed to ensure the requested solution accuracy.

NekRS Strong Scaling, $E = 64^3$, $N = 8$, $n = EN^3 = 134,217,728$, Δt=5.e-3, Δt^*=6.25e-2, CFL=2.42, Ω=[0,4]^3

node	gpu	E	E/gpu	n/gpu	v_i	p_i	T_i	t_{step}	R	R$_{\text{ideal}}$	P$_{\text{eff}}$	r_t
4	24	2.6214e+05	1.0923e+04	5.5924e+06	2	1.81	1	2.44e-01	1.00	1.00	100	3.90
5	30	2.6214e+05	8.7381e+03	4.4739e+06	2	1.86	1	2.06e-01	1.18	1.25	94	3.30
6	36	2.6214e+05	7.2818e+03	3.7283e+06	2	1.81	1	1.80e-01	1.35	1.50	90	2.89
7	42	2.6214e+05	6.2415e+03	3.1957e+06	2	1.88	1	1.61e-01	1.51	1.75	86	2.57
8	48	2.6214e+05	5.4613e+03	2.7962e+06	2	1.82	1	1.39e-01	1.75	2.00	87	2.22
9	54	2.6214e+05	4.8545e+03	2.4855e+06	2	1.82	1	1.31e-01	1.86	2.25	82	2.09
10	60	2.6214e+05	4.3691e+03	2.2370e+06	2	1.87	1	1.20e-01	2.02	2.50	80	1.93
11	66	2.6214e+05	3.9719e+03	2.0336e+06	2	1.85	1	1.11e-01	2.19	2.75	79	1.78
12	72	2.6214e+05	3.6409e+03	1.8641e+06	2	1.89	1	1.05e-01	2.31	3.00	77	1.69
13	78	2.6214e+05	3.3608e+03	1.7207e+06	2	1.85	1	1.01e-01	2.41	3.25	74	1.62
15	90	2.6214e+05	2.9127e+03	1.4913e+06	2	1.89	1	9.37e-02	2.60	3.75	69	1.50
16	96	2.6214e+05	2.7307e+03	1.3981e+06	2	1.90	1	8.66e-02	2.82	4.00	70	1.38

NekRS Strong Scaling, $E = 128^3$, $N = 8$, $n = EN^3 = 1,073,741,824$, Δt=2.e-3, Δt^*=2.5e-2, CFL=1.78, Ω=[0,4]^3

node	gpu	E	E/gpu	n/gpu	v_i	p_i	T_i	t_{step}	R	R$_{\text{ideal}}$	P$_{\text{eff}}$	r_t
32	192	2.0972e+06	1.0923e+04	5.5924e+06	1	1.14	1	2.24e-01	1.00	1.00	100	8.96
40	240	2.0972e+06	8.7381e+03	4.4739e+06	2	1.19	1	2.01e-01	1.11	1.25	89	8.04
50	300	2.0972e+06	6.9905e+03	3.5791e+06	2	1.18	1	1.65e-01	1.35	1.56	86	6.60
60	360	2.0972e+06	5.8254e+03	2.9826e+06	2	1.17	1	1.45e-01	1.54	1.87	82	5.80
70	420	2.0972e+06	4.9932e+03	2.5565e+06	2	1.16	1	1.26e-01	1.77	2.18	81	5.04
80	480	2.0972e+06	4.3691e+03	2.2370e+06	2	1.18	1	1.14e-01	1.96	2.50	78	4.56
90	540	2.0972e+06	3.8836e+03	1.9884e+06	2	1.19	1	1.05e-01	2.13	2.81	75	4.20
100	600	2.0972e+06	3.4953e+03	1.7896e+06	2	1.16	1	9.92e-02	2.25	3.12	72	3.96

NekRS Strong Scaling, $E = 256^3$, $N = 8$, $n = EN^3 = 8,589,934,592$, Δt=1.e-3, Δt^*=1.25e-2, CFL=1.75, Ω=[0,4]^3

node	gpu	E	E/gpu	n/gpu	v_i	p_i	T_i	t_{step}	R	R$_{\text{ideal}}$	P$_{\text{eff}}$	r_t
256	1536	1.6777e+07	1.0923e+04	5.5924e+06		-		-	-	-	-	
320	1920	1.6777e+07	8.7381e+03	4.4739e+06	1	1.16	1	2.10e-01	1.00	1.00	100	16.8
400	2400	1.6777e+07	6.9905e+03	3.5791e+06	1	1.20	1	1.80e-01	1.16	1.25	93	14.4
480	2880	1.6777e+07	5.8254e+03	2.9826e+06	1	1.22	1	1.54e-01	1.36	1.50	91	12.3
640	3840	1.6777e+07	4.3691e+03	2.2370e+06	1	1.25	1	1.28e-01	1.63	2.00	81	10.3
800	4800	1.6777e+07	3.4953e+03	1.7896e+06	1	1.18	1	1.05e-01	2.00	2.50	80	8.4

Table 13: NekRS *strong scaling* performance on Summit for varying resolution on a fixed domain $\Omega = [0, 4]^3$. Restarting with initial condition at time= 6 hours, performed for 200 steps and the averaged timing per step, t_{step}, is measured in seconds for 101–200 steps with the timestep size Δt in nondimension and Δt^* in physical unit. Ω represents the computational domain in nondimension. The real time ratio is $r_t = t_{step}/\Delta t^*$ where $\Delta t^* = (L/u)\Delta t$ with the characteristic length, L=100m, and the stream velocity, 8m/s.

Figure 29 provides a graphical depiction of the steps that the developed work flow supports.

As outlined in [36] the process of construction of the analysis geometry involves applying a set of tools that first obtains the geometry of the system components in terms of CAD and other models, and performs geometric modeling operations to modify and combine those. To support geometry operations multiple tools have been integrated into this workflow include geometric modelers, Open Cascade [30] and SimModeler [37], that are capable of providing the non-manifold geometric models needed to support fully automatic mesh generation, and user interfaces tools, PetraM [35, 34] and SimModeler UI to support the interactive operations required to modify and combine geometry.

SimModeler was used in the analysis geometry construction operations in the example shown in Figure 29. The upper left image shows a portion of an antenna array as defined in the CAD system and the analysis models created by the application of selected SimModeler geometric simplification operations. The upper right image depicts the discrete data imported that represents two flux surfaces used in the construction of the complete analysis model. The left image in the middle row shows the final analysis geometry that includes the tokamak walls, the flux surfaces and the antenna array. The analysis attributes can be applied in either SimModeler or PetraM. The right image in the middle row of figure 29 shows the use of the PetraM user interface for attribute specification. Mesh control information is also specified directly on the analysis

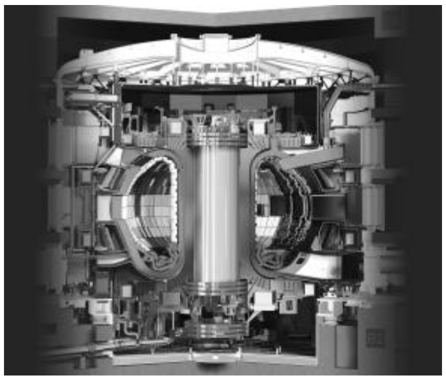
NekRS Weak Scaling, $N = 8$, Δt=4.e-3, Δt^*=5.e-2(sec), CFL=1.93, $r_t = t_{step}/\Delta t^*$, CHAR (1 substep)													
node	gpu	E	Ω	E/gpu	n/gpu	v_i	p_i	T_i	t_{step}	R	$\mathrm{R_{ideal}}$	$\mathrm{P_{eff}}$	r_t
5	30	$64^2 \times 64$	$[0, 4]^2 \times [0,4]$	8738	4.4739e+06	1	1.5	1	0.191	1.00	1.00	100	3.8
20	120	$128^2 \times 64$	$[0, 8]^2 \times [0,4]$	8738	4.4739e+06	1	1.7	1	0.200	0.95	1.00	95	4.0
80	480	$256^2 \times 64$	$[0,16]^2 \times [0,4]$	8738	4.4739e+06	1	1.9	1	0.218	0.87	1.00	87	3.8
320	1920	$512^2 \times 64$	$[0,32]^2 \times [0,4]$	8738	4.4739e+06	1	2.4	1	0.235	0.81	1.00	81	4.7

NekRS Weak Scaling, $N = 8$, Δt=8.e-3, Δt^*=10.e-2(sec), CFL= 3.85, $r_t = t_{step}/\Delta t^*$, CHAR (2 substep)													
node	gpu	E	Ω	E/gpu	n/gpu	v_i	p_i	T_i	t_{step}	R	$\mathrm{R_{ideal}}$	$\mathrm{P_{eff}}$	r_t
5	30	$64^2 \times 64$	$[0, 4]^2 \times [0,4]$	8738	4.4739e+06	2	2.9	1	0.321	1.00	1.00	100	3.2
20	120	$128^2 \times 64$	$[0, 8]^2 \times [0,4]$	8738	4.4739e+06	2	3.1	1	0.335	0.95	1.00	95	3.2
80	480	$256^2 \times 64$	$[0,16]^2 \times [0,4]$	8738	4.4739e+06	2	3.1	1	0.354	0.91	1.00	91	3.5
320	1920	$512^2 \times 64$	$[0,32]^2 \times [0,4]$	8738	4.4739e+06	2	3.4	1	0.376	0.85	1.00	85	3.7

NekRS: Weak Scaling, $N = 8$, Δt=4.e-3, Δt^*=5.e-2(sec), CFL=1.93, $r_t = t_{step}/\Delta t^*$, CHAR (1 substep)													
node	gpu	E	Ω	E/gpu	n/gpu	v_i	p_i	T_i	t_{step}	R	$\mathrm{R_{ideal}}$	$\mathrm{P_{eff}}$	r_t
10	60	$64^2 \times 64$	$[0, 4]^2 \times [0,4]$	4369	2.2370e+06	1	1.5	1	0.112	1.00	1.00	100	2.2
40	240	$128^2 \times 64$	$[0, 8]^2 \times [0,4]$	4369	2.2370e+06	1	1.7	1	0.118	0.95	1.00	95	2.4
160	960	$256^2 \times 64$	$[0,16]^2 \times [0,4]$	4369	2.2370e+06	1	2.1	1	0.127	0.88	1.00	88	2.5
640	3840	$512^2 \times 64$	$[0,32]^2 \times [0,4]$	4369	2.2370e+06	1	2.5	1	0.147	0.76	1.00	76	2.9

NekRS Weak Scaling, $N = 8$, Δt=8.e-3, Δt^*=10.e-2 (sec), CFL= 3.85, $r_t = t_{step}/\Delta t^*$, CHAR (2 substep)													
node	gpu	E	Ω	E/gpu	n/gpu	v_i	p_i	T_i	t_{step}	R	$\mathrm{R_{ideal}}$	$\mathrm{P_{eff}}$	r_t
10	30	$64^2 \times 64$	$[0, 4]^2 \times [0,4]$	4369	2.2370e+06	2	2.9	1	0.186	1.00	1.00	100	1.8
40	240	$128^2 \times 64$	$[0, 8]^2 \times [0,4]$	4369	2.2370e+06	2	3.0	1	0.192	0.96	1.00	96	1.9
160	960	$256^2 \times 64$	$[0,16]^2 \times [0,4]$	4369	2.2370e+06	2	3.1	1	0.216	0.86	1.00	86	2.1
640	3840	$512^2 \times 64$	$[0,32]^2 \times [0,4]$	4369	2.2370e+06	2	3.4	1	0.218	0.85	1.00	85	2.1

Table 14: NekRS *weak scaling* performance on Summit for varying sizes of the domain Ω with a fixed mesh density and resolution per gpu. Restarting with initial condition at time= 6 hours, performed for 200 steps and the averaged timing per step, t_{step}, is measured in seconds for 101–200 steps with the timestep size Δt in nondimension and Δt^* in physical unit. Ω represents the computational domain in nondimension. The real time ratio is $r_t = t_{step}/\Delta t^*$ where $\Delta t^* = (L/u)\Delta t$ with the characteristic length, L=100m, and the stream velocity, 8m/s. Note that n/gpu for 4.4M and 2.2M project roughly 89-95% and 80% parallel efficiency, respectively.



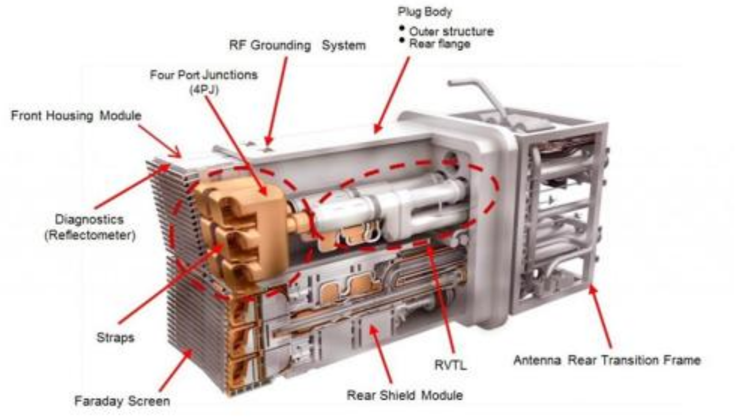


Figure 28: Left: A cutaway view of the ITER tokamak. Right: One of the RF antenna assembles (copied with permission from the ITER Organization web page https://www.iter.org).

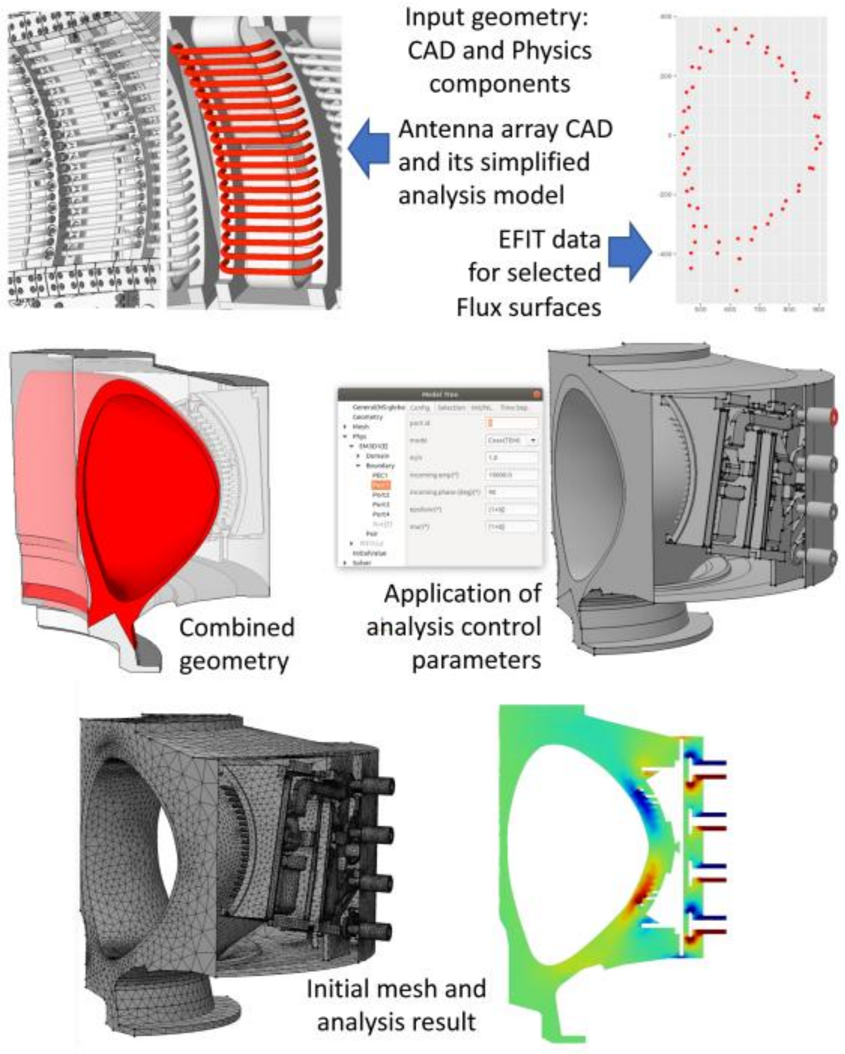


Figure 29: Key steps in the workflow for RF simulations of tokamak fusion Systems

model entities. From there the mesh (lower left image) is automatically generated. Both MeshSim [37] and Gmsh [17] automatic mesh generation procedures are supported.

At this point the information needed for the finite element analysis is complete and the input to the finite element analysis can be provided via input files or using in-memory data transfer methods [18, 38]. The MFEM high-order finite element library [2, 28] was selected by the Center for Integrated Simulation of Fusion Relevant RF Actuators to provide the core of the finite element analysis capability used in the developed simulation workflow. The key reasons for the selection of MFEM include:

- Support of high-order finite elements that provide high rates of convergence that can be effectively taken advantage of when solving wave types of equations. Resent results [33] on RF test cases have clearly shown the advantage of high order methods.
- Emphasis on performant and scalable execution on the up-coming exascale computers for which almost all of their raw compute capability will be provided by accelerators with a current emphasis on GPUs.
- Ready support of the full range of weak equation forms allowing the easy development of finite elements that address RF physics and plasma physics models. The Center for Integrated Simulation of Fusion

Relevant RF Actuators is also using MFEM to develop prototypes analysis capabilities address additional physics modeling needs.

- Ease of extensibility to include new capabilities and to couple with other components that execute additional functions. This extensibility is critical to our ability to integrate the conforming mesh adaptation components being used.

Once an analysis step is performed, the error indication procedure described below is executed. The local error contributions are used to estimate the desired mesh sizes over the domain. This mesh size information is then used by the mesh adaptation procedure that is responsible for modifying the mesh to satisfy the requested mesh sizes. Given that the initial mesh is a graded mesh of simplex elements over geometrically complex domains, and the reliable modeling of wave like problems requires the initial meshes be sufficiently fine, a mesh adaptation procedure that can perform general curved mesh modifications, including coarsening past the initial mesh, and that can create geometry following mesh anisotropy are needed. The PUMI based conforming mesh adaptation procedures [20, 23, 25] can meet these requirements and are being used in this simulation workflow.

Mesh discretization errors in finite element simulations are a result of using finite dimensional solution spaces. In this work we make use of the recovery-based *a posteriori* procedure developed in [42], specialized for the Maxwell's equation discretizations used in MFEM.

The elemental errors for the electric field can be defined as

$$\|e\|_{\Omega_e} := \left[\int_{\Omega_e} (\mathbf{E} - \mathbf{E}_h)^2 \, d\Omega\right]^{1/2} \tag{1}$$

where $\mathbf{E}$ and $\mathbf{E}_h$, are the exact and finite element approximation for the electric field, respectively. Since the exact solution is unknown, one can replace it with an improved approximation of the solution ($\mathbf{E}^*$) to obtain an estimate for the discretization error. In this work the improved approximation is generated using a patch recovery scheme that employs a least-squares fit to create a C^0 field for a C^{-1} discontinuous solution field of interest. MFEM's discretization of the Maxwell's equation represents the electric field with Nedelec basis for which the normal components are discontinuous at element boundaries. Thus given the finite element solution $\mathbf{E}_h$, we sample this field at integration points in a patch of element around each node and fit a continuous polynomial of a given order to obtain the recovered field $\mathbf{E}^*$ at that node.

With this we obtain the following approximation for the element error

$$\|e\|_{\Omega_e} \approx \|e^*\|_{\Omega_e} := \left[\int_{\Omega_e} (\mathbf{E}^* - \mathbf{E}_h)^2 \, d\Omega\right]^{1/2} \tag{2}$$

The total error over, as well as the percentage error, can then be defined as follows

$$\|e\|_{L_2} \approx \left(\sum_{e=1}^{n} \|e^*\|^2_{\Omega_e}\right)^{1/2} \quad \text{and} \quad \eta := \frac{\|e\|_{L_2}}{\|\mathbf{E}^*\|_{L_2}} \tag{3}$$

where n is the total number of elements in the mesh.

A general requirement is then to have the percentage error below a certain threshold, i.e.,

$$\eta \leq \eta_{max}. \tag{4}$$

Furthermore, the error contribution in individual elements in an optimal mesh are expected to be approximately equal. Using these criteria we can then define the desired mesh size for each element in the mesh as follows

$$h_{desired} = h_{current} \times r_e, \tag{5}$$

where $h_{current}$ is the current size of the element and r_e is the element size factor defined as follows

$$r_e = \|e^*\|_{\Omega_e}^{-\frac{2}{2p+d}} \left(\frac{\eta^2_{max}\,\|\mathbf{E}^*\|^2_{L_2}}{\sum_{e'=1}^{n} \|e^*\|_{\Omega_{e'}}^{\frac{2d}{2p+d}}}\right)^{\frac{1}{2p}} \tag{6}$$

where d is the dimension of the mesh and p is the polynomial order of the finite element basis.

We apply the adaptive workflow to full wave simulation of *lower hybrid* waves in ITER plasmas as described in [26, 6] and references therein. Figure 30 shows the problem domain. We will solve axis-symmetric Maxwell's equations with out-of-plane wave number of 200 in the vacuum and plasma regions, and 0 in the waveguides. The in-plane components of the electric field (E_r and E_z) are discretized using high order

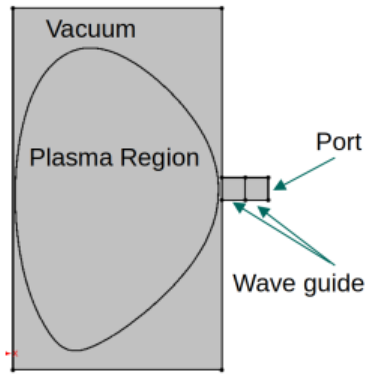


Figure 30: Setup for the axis-symmetric full wave simulation of lower hybrid waves in ITER plasmas.

Nedelec finite element basis, while the out-of-plane component (E_ϕ) is discretized using high-order nodal finite element basis. The result presented here employ quadratic elements.

Figure 31 shows the meshes and solutions for the starting mesh, as well as, three successive adapted meshes. As can be seen in these figures, the solution on the starting coarse mesh is not resolved at all. Figure 32 shows the zoomed version of the initial and final meshes/solutions. As seen in the figures the mesh adaption process nicely resolves the fine scale solution features.

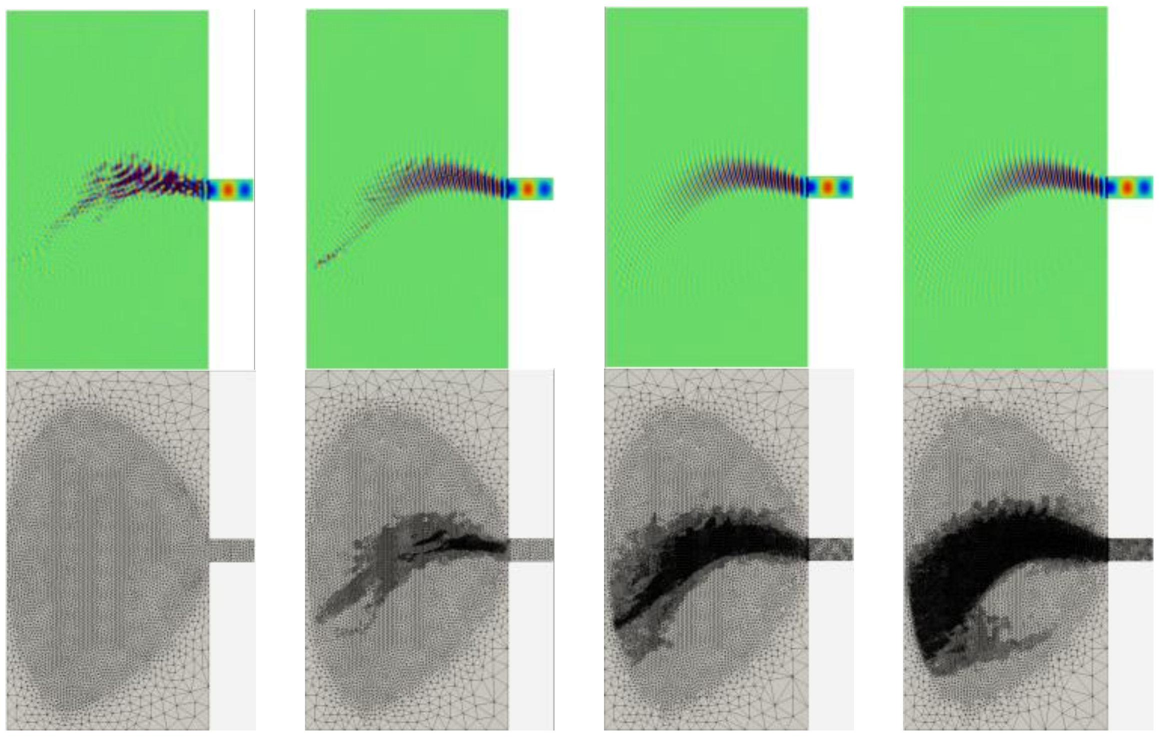


Figure 31: Left to right: the initial (a) and adapted (b-d) meshes with corresponding solution fields.

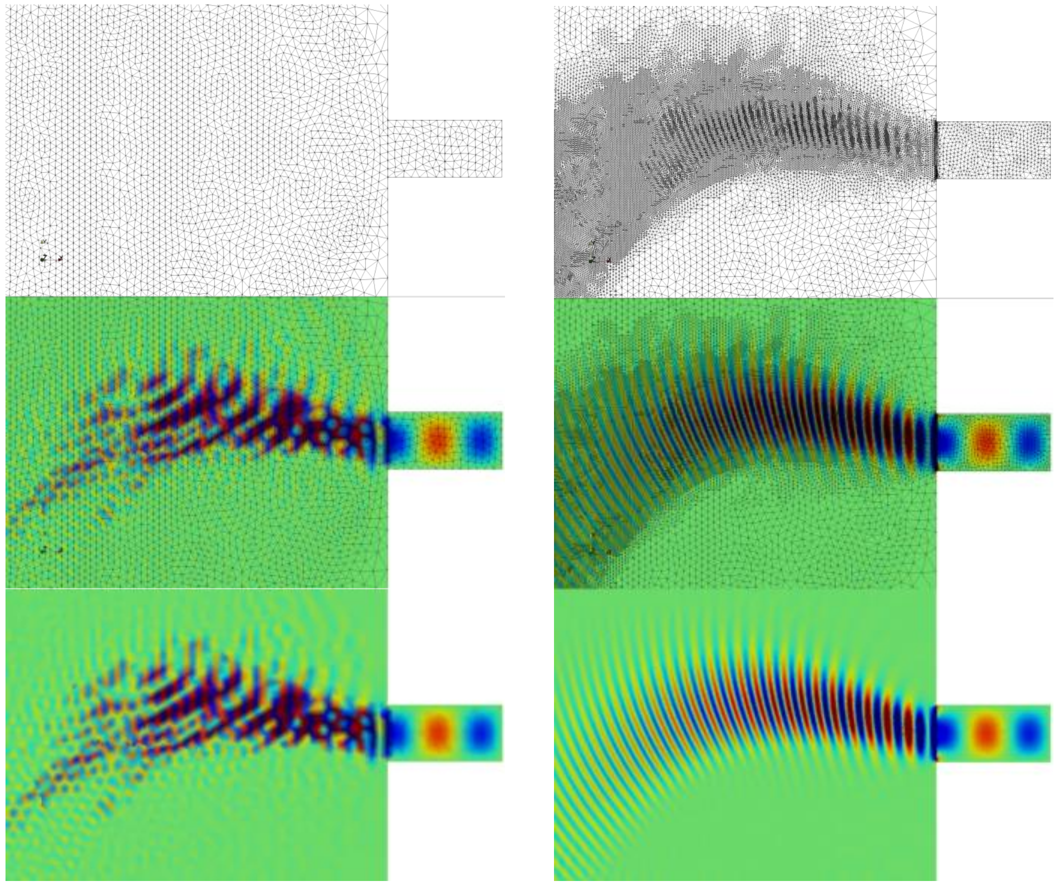


Figure 32: Close-ups of the initial and final adapted meshes and solution fields from Figure 31.

In the second example we consider a 3D simulation of an ICRF antenna model of the Large Plasma Device (LAPD). This example consists of a single strap antenna Figure 33 in a cylinder. The plasma region has an anisotropic permittivity which varies radially. The input current is applied as a port boundary condition at the *coax* port shown in the picture. Figure 34 shows the initial and adapted meshes and the solution (phase)

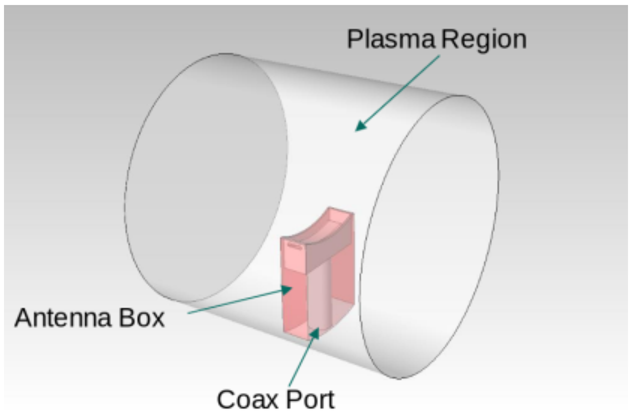


Figure 33: Setup for the 3D LAPD example.

fields. As can be seen in these figure the solution field is not properly resolved on the initial coarse mesh.

However, as the mesh adapts the slow wave region of the plasma domain, the solution features are resolved.

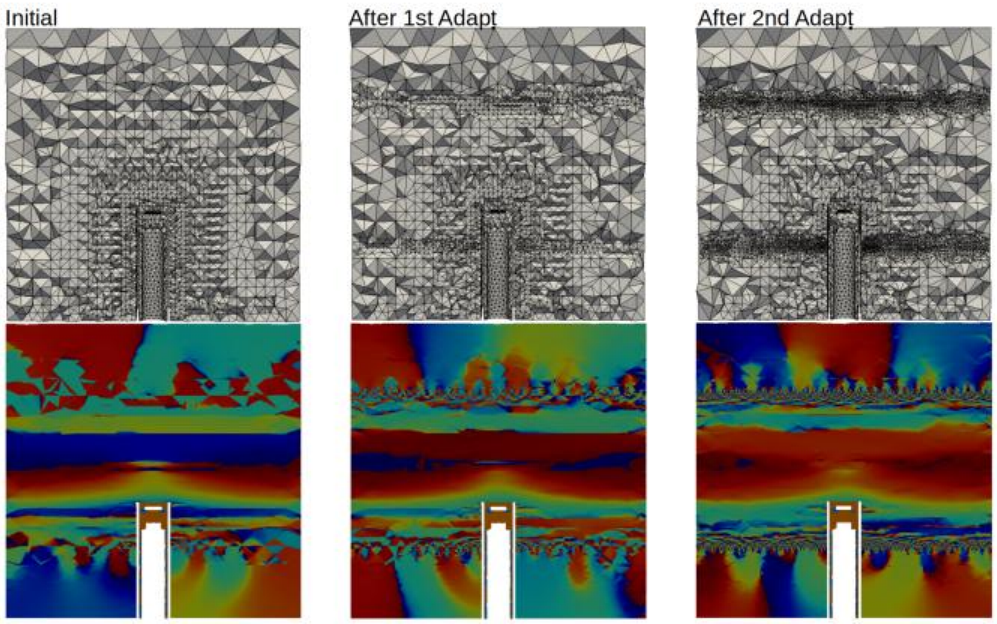


Figure 34: Initial and adapted meshes and solution fields for the LAPD example.

7. OTHER PROJECT ACTIVITIES

7.1 Performance testing on Fugaku

Several groups in CEED have been working on porting and performance testing on the Fugaku system at RIKEN, Japan as part of collaboration between the DOE and the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT). The Nek team also was awarded access to half of the Fugaku system as part of a MEXT call, which enabled large-scale science simulation. Only seven teams were awarded such time and Nek was the only US-based team in that group. Some of the results from this work are reported in Section 3.

7.2 CEED minisymposiums at SIAM CSE21

Ten CEED researchers participated in the SIAM CSE21 conference, where CEED organized two minisymposiums (16 talks total) on "Exascale Software and Algorithms for High-Order PDE Solvers" and "Exascale Applications for High-Order PDE Solvers". CEED researchers also participated in various additional minisymposiums related to matrix free preconditioning, DG methods, agile software development, and more.

7.3 CEED researchers awarded INCITE grant

Several CEED researchers were part of a recently awarded INCITE grant [39] for *Direct Numerical Simulation of Separated Flow over a Speed Bump at Higher Reynolds Numbers.*

7.4 Outreach

CEED researchers were involved in a number of outreach activities, including: participation in the ASCR workshop on "Reimagining Co-Design"; presentations at AMD research, CEED annual review, Naval Nuclear Lab, and the UK Excalibur SLE workshop "Towards Exascale Simulation of integrated engineering systems at extreme scale"; review committees for the SC21 workshop program and ATPESC 2021; several papers;

and updated CEED page on the ECP website. The CEED bake-off paper [14], was also featured on the ECP website under the heading *CEED project examines scalability and performance of key algorithms for HPC applications.*

8. CONCLUSION

In this milestone we improved the high-order software ecosystem for CEED-enabled ECP applications by making progress on efficient matrix-free kernels targeting forthcoming ECP architectures. These kernels included matrix-free preconditioning and the development of new set of CEED solver bake-off problems.

As part of this milestone, we also released the next version of the CEED software stack, CEED-4.0, reported on results from several application collaborations and documented the efforts of porting to AMD GPUs for Frontier and other modern architectures, such as Fugaku.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation's exascale computing imperative.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, LLNL-TR-821011.

This research used computational resources of the supercomputer Fugaku provided by the RIKEN Center for Computational Science.

The research used resources of the Argonne Leadership Computing Facility, which is supported by the U.S. Department of Energy, Office of Science, under Contract DE-AC02-06CH11357. This research also used resources of the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract DE-AC05-00OR22725. Support was also given by the Frontier Center of Excellence.

REFERENCES

[1] Mark Ainsworth and J Tinsley Oden. A posteriori error estimation in finite element analysis. *Computer methods in applied mechanics and engineering*, 142(1-2):1–88, 1997.

[2] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio Kolev, et al. MFEM: a modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021.

[3] Mario Arioli. A stopping criterion for the conjugate gradient algorithm in a finite element method framework. *Numerische Mathematik*, 97(1):1–24, 2004.

[4] Anthony P Austin, Noel Chalmers, and Tim Warburton. Initial guesses for sequences of linear systems in a gpu-accelerated incompressible flow solver. *arXiv preprint arXiv:2009.10863*, 2020.

[5] P. Bello-Maldonado and P.F. Fischer. Scalable low-order finite element preconditioners for high-order spectral element Poisson solvers. *SIAM J. Sci. Comput.*, 41:S2–S18, 2019.

[6] N Bertelli, EJ Valeo, DL Green, M Gorelenkova, CK Phillips, M Podestà, JP Lee, JC Wright, and EF Jaeger. Full-wave simulations of icrf heating regimes in toroidal plasma with non-maxwellian distribution functions. *Nuclear Fusion*, 57(5):056035, 2017.

[7] Paul Bonoli. Center for integrated simulation of fusion relevant RF actuators, 2020.

[8] Jed Brown, Veselin Dobrev, Som Dutta, Paul Fischer, Kazem Kamran, Tzanio Kolev, David Medina, Misun Min, Thilina Ratnayaka, Mark Shephard, Cameron Smith, and Jeremy Thompson. CEED ECP Milestone Report: Propose high-order mesh/data format, June, 2018.

[9] C. Canuto, P. Gervasio, and A. Quarteroni. Finite-element preconditioning of G-NI spectral methods. *SIAM J. Sci. Comput.*, 31:4422–44251, 2010.

[10] N. Chalmers, A. Karakus, A. P. Austin, K. Swirydowicz, and T. Warburton. libParanumal: a performance portable high-order finite element library, 2020. Release 0.3.1.

[11] Denis Davydov, Jean-Paul Pelteret, Daniel Arndt, Martin Kronbichler, and Paul Steinmann. A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid. *International Journal for Numerical Methods in Engineering*, 121(13):2874–2895, 2020.

[12] Veselin Dobrev, Jack Dongarra, Jed Brown, Paul Fischer, Azzam Haidar, Ian Karlin, Tzanio Kolev, Misun Min, Tim Moon, Thilina Ratnayaka, Stanimire Tomov, and Vladimir Tomov. ECP Milestone Report CEED-MS6: Identify initial kernels, bake-off problems (benchmarks) and miniapps, July, 2017.

[13] P. Fischer, K. Heisey, and M. Min. Scaling limits for PDE-based simulation (invited). In *22nd AIAA Computational Fluid Dynamics Conference, AIAA Aviation*. AIAA 2015-3049, 2015.

[14] Paul Fischer, Misun Min, Thilina Rathnayake, Som Dutta, Tzanio Kolev, Veselin Dobrev, Jean-Sylvain Camier, Martin Kronbichler, Tim Warburton, Kasia Swirydowicz, and Jed Brown. Scalability of high-performance PDE solvers. *IJHPCA*, 34, 5:562–586, 2020.

[15] P.F. Fischer and J.W. Lottes. Hybrid Schwarz-multigrid methods for the spectral element method: Extensions to Navier-Stokes. In R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, and J. Xu, editors, *Domain Decomposition Methods in Science and Engineering Series*. Springer, Berlin, 2004.

[16] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems on Concurrent Processors*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[17] Gmsh Team. Gmsh web page. `https://gmsh.info/`.

[18] William F Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, ..., and Scott Klasky. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020.

[19] W. Heinrichs. Line relaxation for spectral multigrid methods. *J. Comput. Phys.*, 77:166–182, 1988.

[20] Daniel A. Ibanez, E. Seegyoung Seol, Cameron W. Smith, and Mark S. Shephard. Pumi: Parallel unstructured mesh infrastructure. *ACM Transactions on Mathematical Software (TOMS)*, 42(3):17, 2016.

[21] David S Kershaw. Differencing of the diffusion equation in lagrangian hydrodynamic codes. *Journal of Computational Physics*, 39(2):375–395, 1981.

[22] Tzanio Kolev, Paul Fischer, Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jed Brown, Jean-Sylvain Camier, Noel Chalmers, Veselin Dobrev, Stefan Kerkemeier, Yu-Hsiang Lan, Elia Merzari, Misun Min, Malachi Phillips, Thilina Ratnayaka, Kris Rowe, Jeremy Thompson, Ananias Tomboulides, Stanimire Tomov, Vladimir Tomov, and Tim Warburton. ECP Milestone Report CEED-MS35: Support CEED-enabled ECP applications in their preparation for Aurora/Frontier, September 30, 2020.

[23] Xiangrong Li, Mark S Shephard, and Mark W Beall. 3d anisotropic mesh adaptation by mesh modification. *Computer methods in applied mechanics and engineering*, 194(48-49):4915–4950, 2005.

[24] J. W. Lottes and P. F. Fischer. Hybrid multigrid/Schwarz algorithms for the spectral element method. *J. Sci. Comput.*, 24:45–78, 2005.

[25] Qiukai Lu, Mark S. Shephard, Saurabh Tendulkar, and Mark W. Beall. Parallel mesh adaptation for high-order finite element methods with curved element geometry. *Engineering with Computers*, 30(2):271–286, 2014.

[26] Orso Meneghini and Syun'ichi Shiraiwa. Full wave simulation of lower hybrid waves in iter plasmas based on the finite element method. *Plasma and Fusion Research*, 5:S2081–S2081, 2010.

[27] Elia Merzari, Haomin Yuan, Misun Min, Dillon Shaver, Ronald Rahaman, Patrick Shriwise, Paul Romano, Alberto Talamo, YuHsiang Lan, Derek Gaston, Richard Martineau, Paul Fischer, and Yassin Hassan. Cardinal: A lower length-scale multiphysics simulator for pebble bed reactors. *Nuclear Technology, American Nuclear Society*, pages 1–23, 2021.

[28] MFEM: Modular finite element methods [Software]. `https://mfem.org`, 2021.

[29] Misun Min, Jed Brown, Veselin Dobrev, Paul Fischer, Tzanio Kolev, David Medina, Elia Merzari, Aleks Obabko, Scott Parker, Ron Rahaman, Stanimire Tomov, Vladimir Tomov, and Tim Warburton. ECP Milestone Report CEED-MS8: Initial Integration of CEED Software in ECP Applications, October, 2017.

[30] Open Cascade community Edition. Open cascade web page. `https://github.com/tpaviot/oce`, 2020.

[31] S.A. Orszag. Spectral methods for problems in complex geometry. *J. Comput. Phys.*, 37:70–92, 1980.

[32] J-F Remacle, Rajesh Gandham, and Tim Warburton. Gpu accelerated spectral finite elements on all-hex meshes. *Journal of Computational Physics*, 324:246–257, 2016.

[33] S Shiraiwa, N Bertelli, M Ono, C Lau, MS Shephard, et al. Recent progress in development of full torus size rf simulation on NSTX-U. In *APS Division of Plasma Physics Meeting Abstracts*, volume 2020, 2020.

[34] S Shiraiwa, T Fredian, J Hillairet, and J Stillerman. πscope: Python based scientific workbench with mdsplus data visualization tool. *Fusion Engineering and Design*, 112:835–838, 2016.

[35] S Shiraiwa, JC Wright, PT Bonoli, T Kolev, and M Stowell. RF wave simulation for cold edge plasmas using the mfem library. In *EPJ Web of Conferences*, volume 157, page 03048. EDP Sciences, 2017.

[36] Morteza H. Siboni and Mark S. Shephard. Adaptive workflow for simulation of RF heaters. *Computer Physics Communications*, 2021. submitted.

[37] Simmetrix. Simmetrix: Enabling simulation-based design. `http://www.simmetrix.com/`.

[38] Cameron W Smith, Brian Granzow, Gerrett Diamond, Daniel Ibanez, Onkar Sahni, Kenneth E Jansen, and Mark S Shephard. In-memory integration of existing software components for parallel adaptive unstructured mesh workflows. *Concurrency and Computation: Practice and Experience*, 30(18):e4510, 2018.

[39] Philippe Spalart, Ramesh Balakrishnan, Aleks Obabko, Misun Min, and Paul Fischer. Direct numerical simulation of separated flow over a speed bump at higher reynolds numbers. 2021.

[40] H Yuan, M. A. Yildiz, E. Merzari, Y. Yu, A. Obabko, G. Botha, G. Busco, Y. A. Hassan, and D. T. Nguyen. Spectral element applications in complex nuclear reactor geometries: Tet-to-hex meshing. *Nuclear Engineering and Design*, 357:110422, 2020.

[41] Junchao Zhang, Jed Brown, Satish Balay, Jacob Faibussowitsch, Matthew Knepley, Oana Marin, Richard Tran Mills, Todd Munson, Barry F. Smith, and Stefano Zampini. The PetscSF scalable communication layer, 2021.

[42] Olgierd Cecil Zienkiewicz and Jian Zhong Zhu. The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, 33(7):1331–1364, 1992.