EXASCALE COMPUTING PROJECT

ECP Milestone Report
Document and popularize CEED-developed software and standards
WBS 2.2.6.06, Milestone CEED-MS41

Tzanio Kolev
Paul Fischer
Ahmad Abdelfattah
Ramesh Balakrishnan
Natalie Beams
Jed Brown
Jean-Sylvain Camier
Hugh Carson
Robert Carson
Noel Chalmers
Matthew Churchfield
Veselin Dobrev
Sebastian Grimberg
Yichen Guo
Stefan Kerkemeier
Yu-Hsiang Lan
Victor A. Mateevitsi
Matthew McCall
Elia Merzari
Misun Min
Ketan Mittal
Will Pazner
Malachi Phillips
Finnur Pind
Thilina Ratnayaka
Robert N. Rieben
Kris Rowe
Mark S. Shephard
Cameron W. Smith
Thomas Stitt
Michael Sprague
Amik St-Cyr
Solvi Thrastarson
Ananias Tomboulides
Stanimire Tomov
Vladimir Tomov
Arturo Vargas
Tim Warburton
Kenneth Weiss

September 30, 2023

ECP Milestone Report
Document and popularize CEED-developed software and standards
WBS 2.2.6.06, Milestone CEED-MS41

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

September 30, 2023

ECP Milestone Report
Document and popularize CEED-developed software and standards
WBS 2.2.6.06, Milestone CEED-MS41

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	September 29, 2023	Original	

EXECUTIVE SUMMARY

The goal of this milestone was to document and popularize the CEED-developed software and standards as part of the completion of the CEED efforts.

In addition to a final report on the CEED work for Frontier, Aurora, and ECP applications, this milestone included developments to engage external applications in the DOE and industry, the public release of the CEED-6.0 software distribution and the next CEED Annual meeting (CEED7AM) which included representatives from ECP applications, vendors and software technology projects. We also summarize the CEED educational impact and report on other project activities like research in stopping criteria, fast coarse grid solves, multi-domain coupling, NVIDIA Hopper performance and in-situ visualization.

The specific tasks addressed in this milestone were as follows.

- CEED-T29 (ADCD04-99): Public release of CEED-6.0
- CEED-T30 (ADCD04-100): Reach out to external applications and sponsors in the DOE and industry
- CEED-T31 (ADCD04-101): CEED Annual meeting (CEED7AM)
- CEED-T32 (ADCD04-102): Final report on all CEED activities, including lessons learned and future outlook

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The goal of this milestone was to document and popularize the CEED-developed software and standards as part of the completion of the CEED efforts.

In addition to a final report on the CEED work for Frontier, Aurora, and ECP applications, this milestone included developments to engage external applications in the DOE and industry, the public release of the CEED-6.0 software distribution and the next CEED Annual meeting (CEED7AM) which included representatives from ECP applications, vendors and software technology projects.

As part of the milestone, we also summarize the CEED educational impact and report on other project activities like research in stopping criteria, fast coarse grid solves, multi-domain coupling, NVIDIA Hopper performance and in-situ visualization.

2. CEED PERFORMANCE IMPROVEMENTS FOR FRONTIER

2.1 NekRS on Frontier

As part of a joint submission with ExaSMR, NekRS was recognized as one of the Gordon Bell Prize finalists for 2023. The submission explored the performance of NekRS on 9000 nodes of Frontier for a multiphysics simulation of a full reactor core.

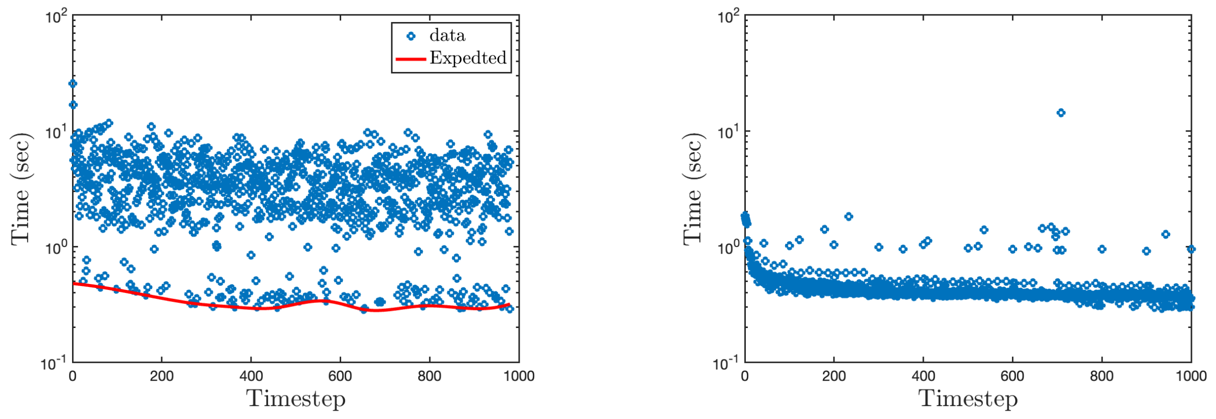


Figure 1: Navier-Stokes time-per-step for conjugate heat transfer (fluid+solid) runs on Frontier: (left) 500-layer case with GPU-direct on 4-15-23, featuring 512 M fluid elements and 586 M solid elements (377 B points total); (right) 800-layer case *without* GPU-direct on 8-6-23, featuring 820 M fluid elements and 938 M solid elements (603 B points total);

Running these billion+ element cases on 72,000 ranks (one per AMD MI250X GCD accelerator) presented several challenges stemming from network noise and platform stability issues that were present only at scale and not manifest at smaller node counts. Figure 1 illustrates the performance challenges faced in initial full-scale production runs. The left plot shows time-per-step for Navier-Stokes runs with 231 B gridpoints on P=72,000 GCDs (two GCDs per AMD MI250X on Frontier). The times break down into a bimodal distribution with the smaller values, highlighted by the red envelope, corresponding to expected values of $\sim$ 0.3–0.4 s/step. Above this group is a large distribution in the 1–10 s/step range, which is clearly spurious (and not due to changes in iteration counts per timestep). An extensive battery of tests pointed to flooding of the network by the innocuous Jacobi-preconditioned velocity solve, which generally is not communication intensive. The velocity solve, however, does put out relatively long messages because all three components are computed simultaneously, which allows one to simultaneously communicate all three components during the gather-scatter operation, thereby reducing message latency overhead. The bimodal nature of the timings is indicative of network congestion, where messages require multiple attempts to get through or are held up through other congestion mitigation techniques. We thus experimented with techniques such as solving for each velocity component separately, thereby putting shorter messages on the network, and avoiding

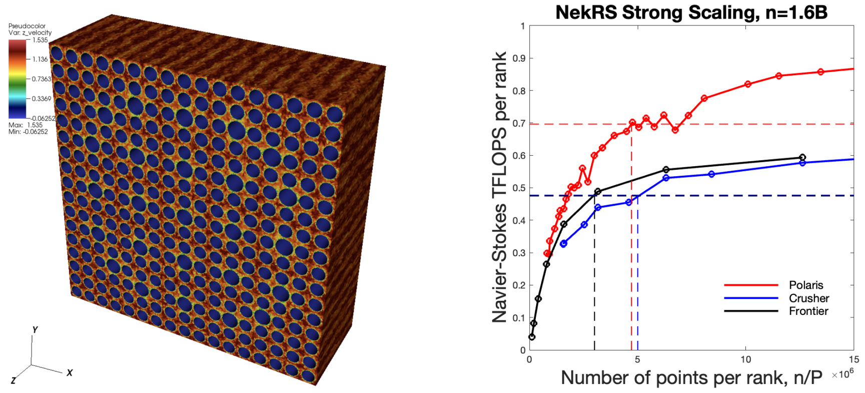


Figure 2: NekRS strong-scaling results showing sustained TFLOPS per rank for Navier-Stokes simulations of a 17×17 rod bundle (illustrated on left) with n=1.6B grid points as a function of n/P for Polaris, Crusher, and Frontier. Horizontal dashed lines are at 80% of the saturated performance. Vertical dashed lines represent $n_{0.8}$.

GPU-direct. Both of these approaches yielded significant noise reduction with only minor increase in the minimal time-per-step. Of these two, *no-GPU-direct* approach was slightly faster and the results in Fig. 1 (right) indicate a significant drop in network-noise overhead. This case sustains 390 GFLOPS per MPI rank for the full Navier-Stokes run.

An central theme within CEED has been to understand the performance parameters that govern time-to-solution on exascale platforms. In this scenario, most of the jobs are not using the full machine. For a given simulation campaign, users will typically pick a number of ranks, P, that is as large as possible while still retaining order-unity parallel efficiency. (We arbitrarily using $\eta = 0.8$ as a target efficiency for purposes of analysis.) Define S_{sat} as the saturated per-rank performance rate for the application (in TFLOPS, say) and the speed on P ranks as $S_P = \eta P S_{sat}$. If the simulation requires W floating point operations (flops) per grid point, then the total work is Wn and the time-to-solution on P ranks is

$$t_\eta \;=\; \frac{W\,n}{S_P} \;=\; \frac{W\,n}{\eta\,PS_{sat}}.$$

For $\eta = 0.8$ one has

$$t_{0.8} \;=\; \frac{W}{0.8}\frac{n_{0.8}}{S_{sat}},$$

which depends crucially on the ratio of $n_{0.8}$ and S_{sat}. Improving node performance (S_{sat}) will reduce time to solution only if the ratio $n_{0.8}/S_{sat}$ does not increase.

We apply the preceding analysis in a series of strong-scale studies with NekRS on Polaris, which is an NVIDIA A100-based platform at Argonne National Laboratory, and on Crusher and Frontier, the AMD MI250X-based machines at Oak Ridge National Laboratory. The scaling results are plotted as TFLOPS per rank versus gridpoints per rank in Fig. 2. In this case, the A100 sustains 870 GFLOPS for $P = 104$, while Crusher and Frontier sustain respectively 593 and 577 GFLOPS for $P = 128$. If we take these values to be the saturated performance levels, S_{sat}, then we find from Fig. 2 that 80% parallel efficiency will be realized at $n_{0.8} := n/P$ values of 4.7 M, 5.0 M, and 3.0 M for Polaris, Crusher, and Frontier, respectively. The respective ratios of $n_{0.8}$ to S_{sat} are 5411, 8383, and 5056, points-per-GFLOPS, which implies that, *at 80% parallel efficiency*, Frontier will have the minimum time to solution. Clearly, at almost every value of $n/P > 1$ M, Polaris outperforms Frontier. However, Frontier strong scales better than Polaris—its efficiency does not fall off as quickly and one finds the rather surprising result that users who obey some relatively strict efficiency rule will be able to run this problem faster on Frontier than on Polaris.

2.2 MAGMA Update: Towards a Nontensor Backend Utilizing AMD's Matrix Cores

The MAGMA backend for libCEED provides nontensor basis actions that rely on standard and customized matrix multiplication (GEMM) kernels. For relatively small order problems, the MAGMA backend provides optimized GPU kernels that perform the `interp` and `grad` basis actions. Those kernels implicitly implement a sequence of on-device GEMM operations, thus improving data reuse over a standard GEMM implementation. However, the MAGMA backend still relies on the standard GEMM routines either from the rocBLAS library or from the MAGMA-BLAS sub-package. The latter option is MAGMA's own standard GEMM implementation, which to this date, does not take advantage of AMD's Matrix Core technology.

The MAGMA development team has been working on utilizing the GEMM hardware accelerators on modern GPUs in order to explore whether the nontensor backend can be further improved for relatively large order problems. While the work addresses both NVIDIA's Tensor Core and AMD's Matrix Core technologies, we focus only on performance results for AMD GPUs in this section. The MAGMA team is addressing a set of challenges to achieve that goal:

- The current goal is to develop a unified kernel codebase that support both NVIDIA and AMD hardware accelerators.
- Not all precisions are accelerated on both architectures. For example, AMD GPUs accelerate single precision GEMM using matrix cores, while NVIDIA GPUs do not.
- Accelerated precisions support hardware-specific dimensions. For example, the NVIDIA A100 GPU supports double precision for $(\bar{m}, \bar{n}, \bar{k}) = (8,\ 8,\ 4)$, while the H100 GPU supports four different combinations $(\bar{m}, \bar{n}, \bar{k}) \in \{(8,\ 8,\ 4),\ (16,\ 8,\ 4),\ (16,\ 8,\ 8),\ (16,\ 8,\ 16)\}$. On the other hand, the AMD MI210 and MI250x GPUs support double precision for sizes $(\bar{m}, \bar{n}, \bar{k}) \in \{(4,\ 4,\ 4),\ (16,\ 16,\ 4)\}$.
- Depending on the precision and the dimensions $(\bar{m}, \bar{n}, \bar{k})$, one or more GEMM operation can be done simultaneously. For example, the MI210 (and MI250x) can perform only one DGEMM operation of size (16, 16, 4), but can perform four simultaneous operation of size (4, 4, 4).
- Each set of $(\bar{m}, \bar{n}, \bar{k})$ requires a specific data layout of the input/output matrices (A, B, and C) in the register file of a single warp.

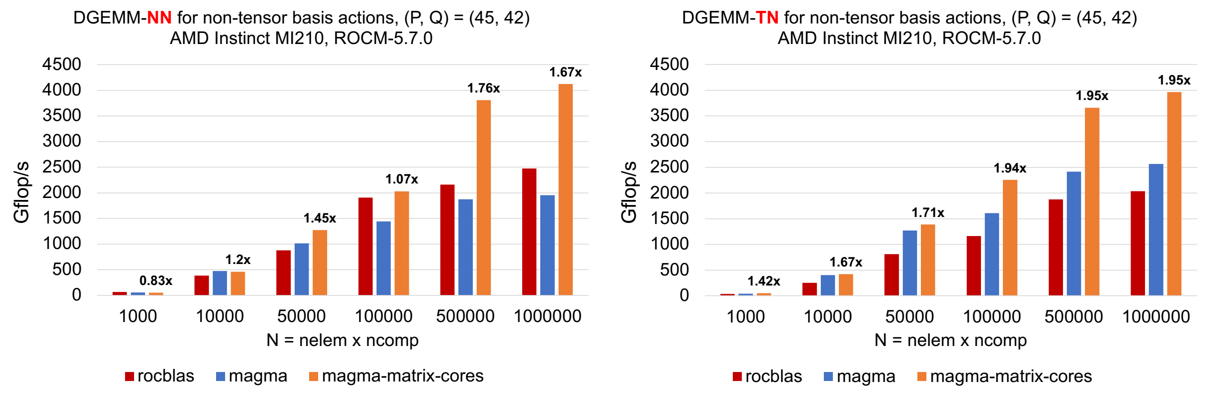


Figure 3: Performance of the DGEMM kernel for the nontensor basis action, (P, Q) = (45, 42). Results are shown for the AMD Instinct MI210 GPU using ROCM-5.7.0. Relative speedups are shown against rocBLAS.

With respect to the above challenges, the MAGMA team has prioritized the design of a unified kernel structure that abstracts all the details of the Tensor Core and Matrix Core technologies. While the design is still in its early stages, the MAGMA team is confident that the design can efficiently support any precision with any set of dimensions $(\bar{m}, \bar{n}, \bar{k})$. The current status of the design can be summarized as follows:

- The kernel design is unified across both NVIDIA and AMD GPUs. The design does not need to change in order to support new precision or new dimensions.

- The kernel is written using C++ templates with up to 13 tuning parameters. These parameters include the precision, the tensor/matrix core dimensions $(\bar{m}, \bar{n}, \bar{k})$, the blocking sizes of the input/output matrices, and the thread configuration.
- The kernel relies on a number of device routines that hide the details of the hardware accelerator. Each device routine has a special template specialization for a specific set of {precision, $(\bar{m}, \bar{n}, \bar{k})$}.
- The developer/user cannot instantiate the kernel for a precision or dimensions that are not supported by the hardware.

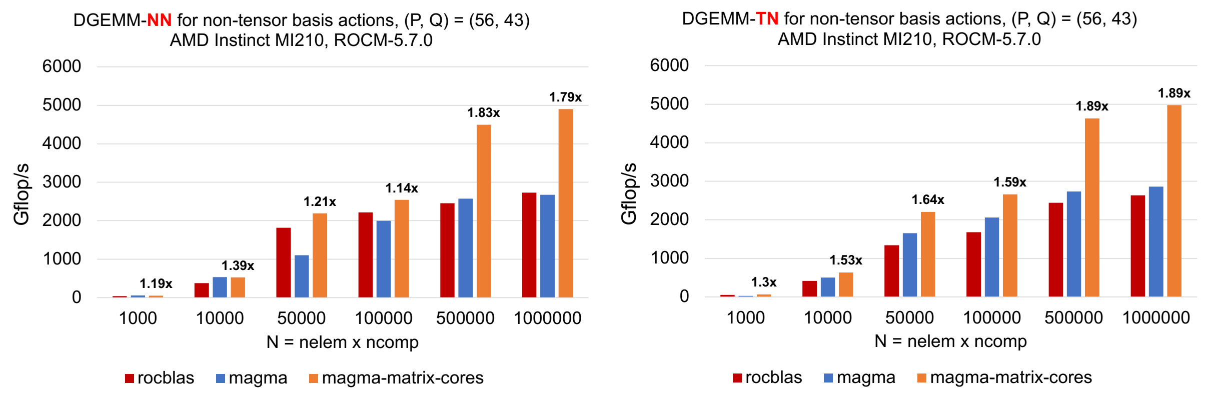


Figure 4: Performance of the DGEMM kernel for the nontensor basis action, (P, Q) = (56, 43). Results are shown for the AMD Instinct MI210 GPU using ROCM-5.7.0. Relative speedups are shown against rocBLAS.

Preliminary Performance Results. We show preliminary results for the DGEMM kernel on the AMD Instinct MI210 GPU, which has the same architecture of the MI250x GPU (`gfx90a`). The details of the conducted experiments are as follows:

- We compare the performance of rocBLAS, MAGMA's latest release (2.7.2), and the ongoing work with MAGMA utilizing the matrix cores
- Each experiment tests a single DGEMM operation with dimensions typically found in the nontensor basis actions of the bakeoff problems.
- The performance results of MAGMA with the matrix core are collected from two kernel instances only. A comprehensive tuning for the new design has not yet been done.
- For each dimension tested, only the matrix A can be transposed. The input matrix B is always non-transposed. It is also short and very wide.
- We consider only dimensions for A that are larger than 40. The specialized kernels currently in the MAGMA nontensor backend can efficiently address smaller dimensions.

Figures 3 through 5 show the performance results for a set of dimension often encountered in libCEED's bake-off problems. A DGEMM-NN operation performs $(C_{P\times N} = \alpha A_{P\times Q} \times B_{Q\times N} + \beta C_{P\times N})$, while a DGEMM-TN operation performs $(C_{Q\times N} = \alpha A^T_{Q\times P} \times B_{P\times N} + \beta C_{Q\times N})$. We observe that, in most cases, the MAGMA kernel utilizing the matrix cores achieves the best performance, with speedups reaching up to 1.95× against rocBLAS. We also observe that utilizing the matrix cores has a huge impact on the performance of MAGMA, with speedups reaching up to 2.5× against MAGMA's latest release.

RocBLAS still has an advantage in some scenarios, but we emphasize that the new MAGMA design has not been thoroughly tested, and so it is possible that a comprehensive tuning sweep would reveal even better performance. In addition, the new design can be used as a foundation for a new set of specialized kernel that perform the nontensor basis actions directly, without calling DGEMM inside a loop.

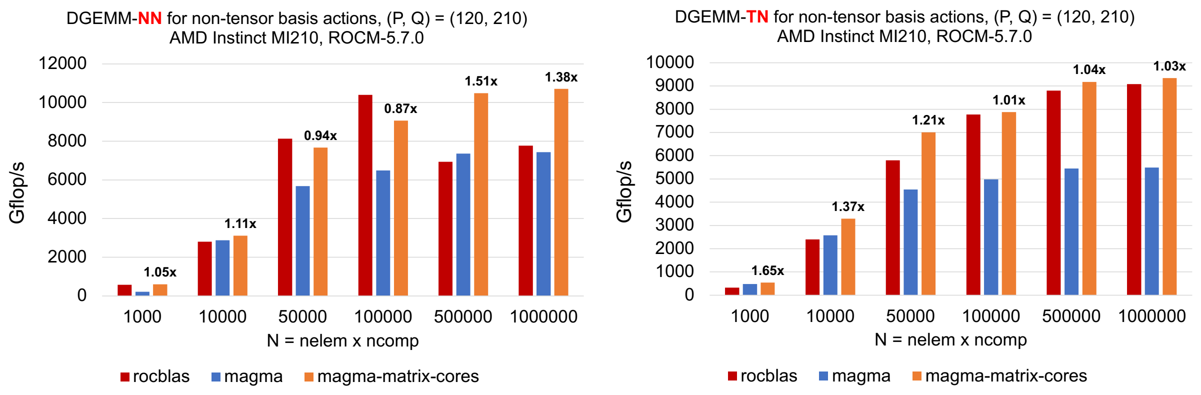


Figure 5: Performance of the DGEMM kernel for the nontensor basis action, (P, Q) = (120, 210). Results are shown for the AMD Instinct MI210 GPU using ROCM-5.7.0. Relative speedups are shown against rocBLAS.

2.3 A Kokkos-based GPU Memory Pool for Omega_h Mesh Adaptation

A memory pool is a technique for managing memory allocation in a computer program. It consists of a pre-allocated block of memory from which the program can request and release memory from the pool as needed, without invoking the system's memory allocator. This can improve the performance, reliability and portability of the program. Some of the benefits of memory pools are:

- They reduce memory fragmentation, which can cause inefficient use of memory and slow down the program.
- They reduce the overhead of system memory allocation and deallocation, which can consume a significant amount of CPU time and introduce latency.
- They allow the programmer to control the size and layout of the memory blocks, which can optimize the memory access patterns and cache efficiency.

Preliminary work done using the CUDA library for unstructured mesh adaptation in Omega_h on NVIDIA GPUs shows a significant performance increase when using a memory pool as opposed to traditional memory management strategies. However, CUDA is a proprietary library developed by NVIDIA for NVIDIA devices. The use of vendor-specific libraries such as CUDA for GPU computing can limit the portability and flexibility of applications. As such, this research aims to achieve comparable performance gains on AMD and Intel devices using the cross-platform library, Kokkos, to implement the memory pool.

Omega_h [55] is a software library written in C++ that provides mesh adaptivity for tetrahedron and triangle meshes, with an emphasis on high-performance computing. It is designed to add adaptive capabilities to existing simulation software. Mesh adaptivity allows for the reduction of both discretization error and the number of degrees of freedom during a simulation, as well as enabling simulations with moving objects and changing geometries. Omega_h achieves this in a manner that is fast, memory-efficient, and portable across a variety of architectures.

There exists a memory pool in Omega_h that works well with the CUDA backend on NVIDIA devices. The memory pool implementation described here instead uses Kokkos [116], a cross-platform library, to obtain device memory. The design for this pool was inspired by Boost's Simple Segregated Storage [32], a fixed-size chunk memory-pool implementation targeting host-sided memory. A fixed-sized chunk design often allows for faster allocation, whereas variably-sized chunk designs, such as those found in Umpire [18], allow for more memory efficiency. An important distinction between Boost's Simple Segregated Storage and other similar host-bound implementations and ours is that the free-list is interweaved into the chunks themselves. While this reduces memory overhead, storing a free-list in device memory would require copying the free-list to the host, manipulating it, and then copying it back to the device each time an allocation or deallocation is performed. As such, this scheme for managing chunks was impractical. It was much more reasonable to store the free-list in host memory. However, now that we have separated the free-list from the underlying chunks pool, we realized it does not have to be a list, which can result in linear allocation time. Thus, we opted to use free-sets instead. Furthermore, since memory is only requested and returned in host-side code, the large parallelism of GPUs does not interfere with traditional set-searching algorithms. As such, we were able to adapt an existing strategy, known to work well for managing host memory, to device memory where it brings significant performance improvements.

In Omega_h, a `StaticKokkosPool` is a non-resizable pool of memory from which an allocation can be made. Upon instantiation of each `StaticKokkosPool`, we call `kokkos_malloc` to get a contiguous block of memory. When we destroy the `StaticKokkosPool`, we call `kokkos_free` to release the memory back to the system. The memory pool is divided into an array of contiguous fixed-size one kiB chunks. Each chunk is ordered and indexed by their position.

Allocation. Instead of using a free-list, which may result in linear allocation time, we use a free-mulitset, `freeSetBySize`, which results in logarithmic allocation time. When an allocation of n bytes is requested, we search through `freeSetBySize` to find the smallest free region that can accommodate the number of requested chunks. The number of requested chunks is calculated by $r = \lceil n \div chunkSizeInBytes \rceil$. In the case of a new or empty `StaticKokkosPools`, `freeSetBySize` would contain only one free region representing the entire pool. Figure 6 shows how `freeSetBySize` relates to the memory in the pool. A free region is defined as a set of contiguous free blocks between allocated regions or ends of the pool. If the found free region has more chunks than the number of requested chunks, then we split the region and only allocate the chunks requested. The remaining region remains in the free list.

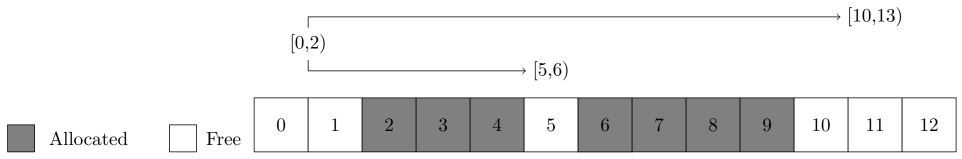


Figure 6: A fragmented pool with `freeSetBySize` showing how free regions are organized in a tree structure. Index pairs are sorted top to bottom by the size of the free region they represent in decreasing order.

Deallocation and Defragmentation. In addition to `freeSetBySize`, `freeSetByIndex` is used to achieve defragmentation in logarithmic time. As the name suggests, index pairs stored in `freeSetByIndex` are sorted in numerical order by the first index as depicted in Figure 7. It is not possible for two index pairs to “overlap” or share the same start or end index.

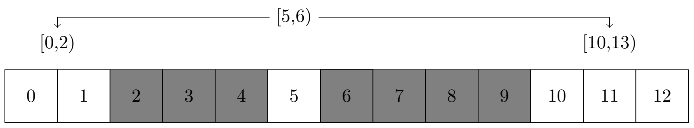


Figure 7: A fragmented pool with `freeSetByIndex` showing how free regions are organized in a tree structure. Index pairs are sorted left to right by the indices of the free region they represent in increasing order.

When memory is returned to the pool, the region is temporarily inserted into the tree as shown in Figure 8.

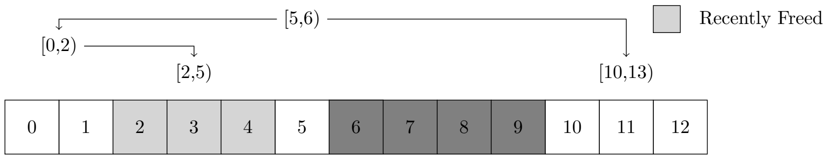


Figure 8: A fragmented pool with `freeSetByIndex` showing with a recently returned region.

We then check for free regions adjacent to the recently returned region, remove the two or three index pairs if any adjacent regions were found, and then insert one index pair encompassing the defragmented region. For example, in Figure 8 the allocation spanning blocks [2,5) is freed. This new free region is surrounded by pre-existing free regions [0,2) and [5,6). Thus we remove the three index pairs and replace them with [0,6). The result is visualized in Figure 9.

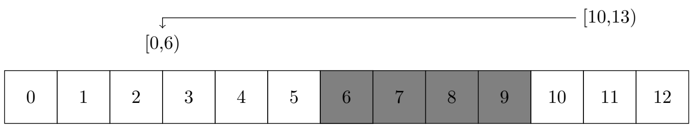


Figure 9: A less fragmented pool with freeSetByIndex shown after defragmentation

Resizing. In the event that we cannot find a suitable free region large enough to satisfy the allocation requested, we make a new StaticKokkosPool. The KokkosPool class performs this for us by maintaining a list of StaticKokkosPools. In Omega_h, KokkosPool is a singleton object that is lazy initialized upon the first allocation request. Upon instantiation, it creates one StaticKokkosPool and adds the pool to its list. Thus, when you allocate through KokkosPool, it begins at the first StaticKokkosPool in the list and tries to allocate from it. If the allocation fails, it moves on to the next StaticKokkosPool and tries to allocated from that. We repeat this for every StaticKokkosPool in the list until we achieve a successful allocation. If we have reached the end of the list and were not able to find a large enough free region in any of the StaticKokkosPools, we allocate a new StaticKokkosPool with the size of $\max(r, mostChunks * g)$, where r is the number of chunks requested as defined above, $mostChunks$ is the number of chunks in the largest StaticKokkosPool, and g is the growth factor. In Omega_h, the default growth factor is two. This strategy ensures an allocation will be successful until the machine runs out of physical memory.

Testing. The implementation discussed here resides in Omega_h, a GPU accelerated mesh adaptivity library. The library was benchmarked with and without the memory pool against the FUN3D delta wing case from the Unstructured Grid Adaptive Working Group adaptation benchmarks [2] on the Oak Ridge Leadership Computing Facility's Frontier. We ran the 500k case, which comprised of 581,196 tetrahedrons before adaptation and 5,283,878 tetrahedrons after adaptation, with the pool disabled and with the Kokkos MemoryEvents tool enabled to get a sense of the allocation patterns for this particular set of benchmarks. We found that it uses at most 670 MiB during the lifetime of the benchmark. Thus we chose an initial pool size of 700 MiB. In addition, we found that allocation requests were greater than one kiB more often than not, thus we decided to set the fixed-chunk size to one kiB.

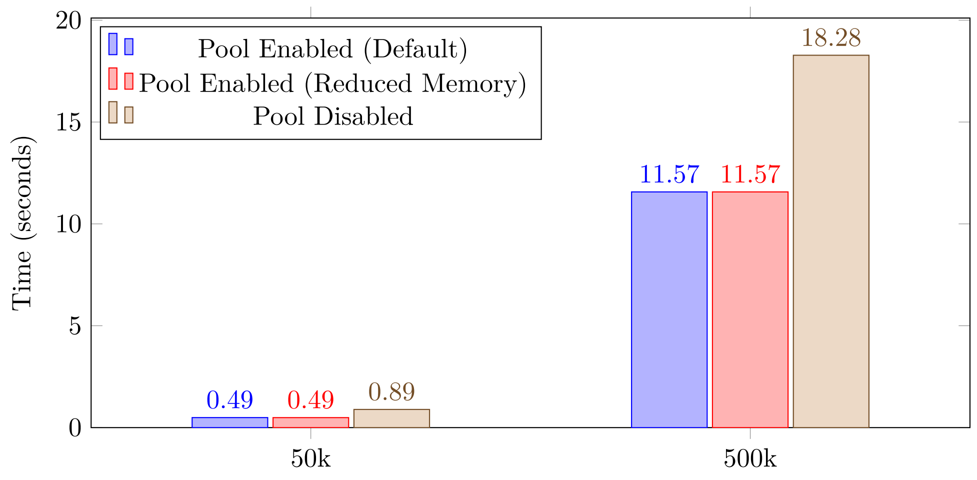


Figure 10: A bar plot showing the performance improvements in the delta wing case brought by the implementation of a memory pool. Less time is better.

Performance. After we determined these parameters for the pool, we ran all subsequent benchmarks without the Kokkos Tools enabled. In the 50k case without pooling, adapting 581,196 tetrahedrons to 533,937 tetrahedrons took 0.89 seconds total. With the pool engaged, this time was reduced to 0.49 seconds, a 45% time reduction. In the 500k case without pooling, adapting 581,196 tetrahedrons to 5,283,878 tetrahedrons5 took 18.28 seconds total while, with the pool enabled, adaptation only took 11.57 seconds on average, a 37% time reduction. We then reduced the initial size of the pool to 100 KiB. This is significantly less than what we determined the benchmarks need and should require the pool to resize more often during the runtime. We found that difference in time reduction brought about by the additional resizing behavior of the pool was less than one percent. Figure 10 shows these results.

This implementation only splits and coalesces free regions, not individual chunks. Thus, it is possible for small allocations, or allocations that don't roughly align with multiples of the chunk size to result in excessive memory waste as the vast majority of a chunk may not be used, especially if chunks are large. Alternative implementations such as Umpire on the other hand split and coalesce individual chunks, resulting in lesser memory usage. Regardless, this implementation manages to bring substantial performance improvements in a cross-platform manner. Thus far, this implementation has been tested on AMD and NVIDIA devices. Future work aims to test this implementation on Intel devices as well. Furthermore, this implementation has only been tested on systems with traditional, separate discrete host and device memory. With the advent of modern unified memory architectures, we intend to test this implementation on newer systems such those that utilize NVIDIA's Grace-Hopper APU and AMD and Intel equivalents. Here, we determined that fixed-size chunk memory pools are just as suitable for use in device memory spaces as host memory spaces. We also determined that such a pool is suitable for use across different devices and vendors. Testing has also confirmed that pooling device memory significantly reduces the running time of an application by up to 45%. In conclusion, fixed-size chunk memory pools are a viable method of managing device memory that brings substantial performance improvements in a cross-platform manner.

2.4 Ratel Plasticity on Tioga

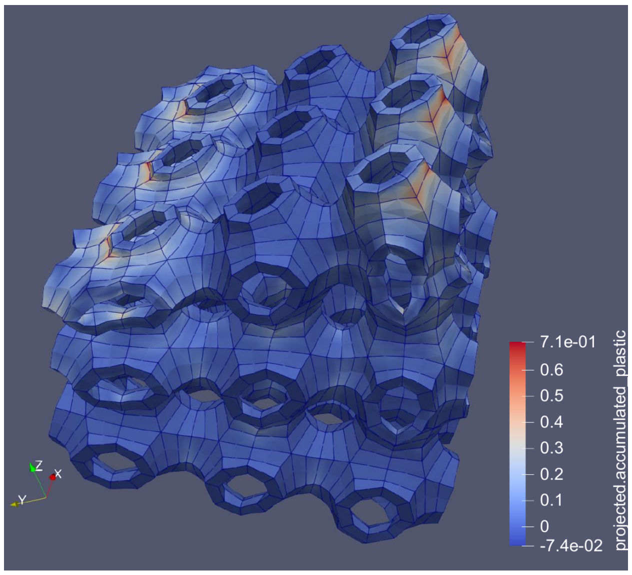


Figure 11: Accumulated plastic strain for an oblique press of a Schwarz Primitive lattice using Ratel.

The Ratel team has developed plasticity models for Ratel using libCEED qfunctions. The formulation includes a return mapping algorithm in the residual evaluation qfunction, with a stored solution to the return mapping problem for use in matrix-free Jacobian application. The plasticity solver has been applied to various benchmark problems as well as nontrivial geometries such as Figure 11 and Figure 12. An efficiency spectrum running increasing problem sizes on one node (8 logical devices of MI250X) and 8 nodes (64 logical

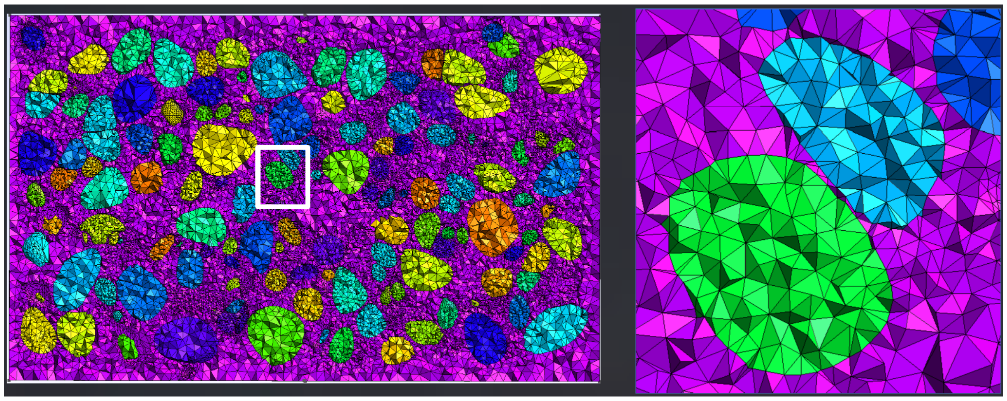


Figure 12: Tetrahedral mesh of grains in binder, segmented from CT scans and meshed using Gmsh.

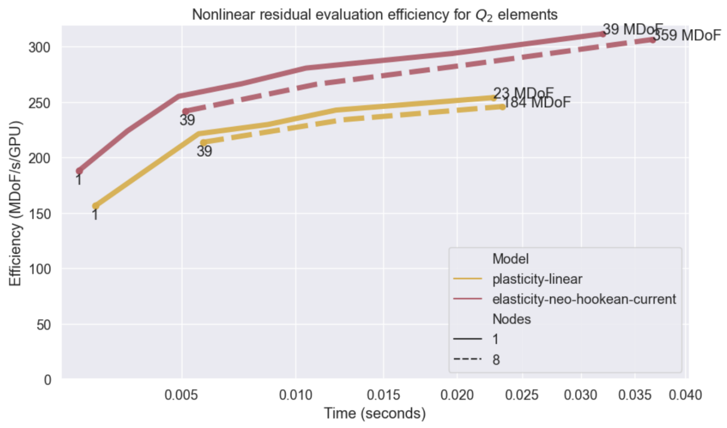


Figure 13: Relative efficiency of hyperelastic and elastoplastic residual evaluation using Ratel on LLNL Tioga's MI250X.

devices) of LLNL's Tioga is shown in Figure 13. We observe that evaluating plasticity residuals incurs only modest efficiency overhead as compared to hyperelasticity and exhibits excellent weak scaling from one to several nodes.

3. CEED PERFORMANCE IMPROVEMENTS FOR AURORA

3.1 NekRS Performance on Aurora and Sunspot

ExaSMR performance for a singlerod on a single PVC.				
System	backend	tiles	$t_{step}(s)$	flops/rank
Aurora	SYCL	1	6.95447e-02	9.01484e+11
Aurora	SYCL	2	4.74475e-02	6.56709e+11
Sunspot	SYCL	1	6.91035e-02	9.02007e+11
Sunspot	SYCL	2	4.82435e-02	6.51462e+11

Table 1: NekRS ExaSMR performance for singlerod simulation on Aurora and Sunspot. t_{step} is measured by averaging timings for 101-500 timesteps for 500 timestep runs. $E = 7168$, $N = 7$ and $n = 2.4M$. $n/P = 2.4$M for one tile and $n/P = 1.2$M for two tiles.

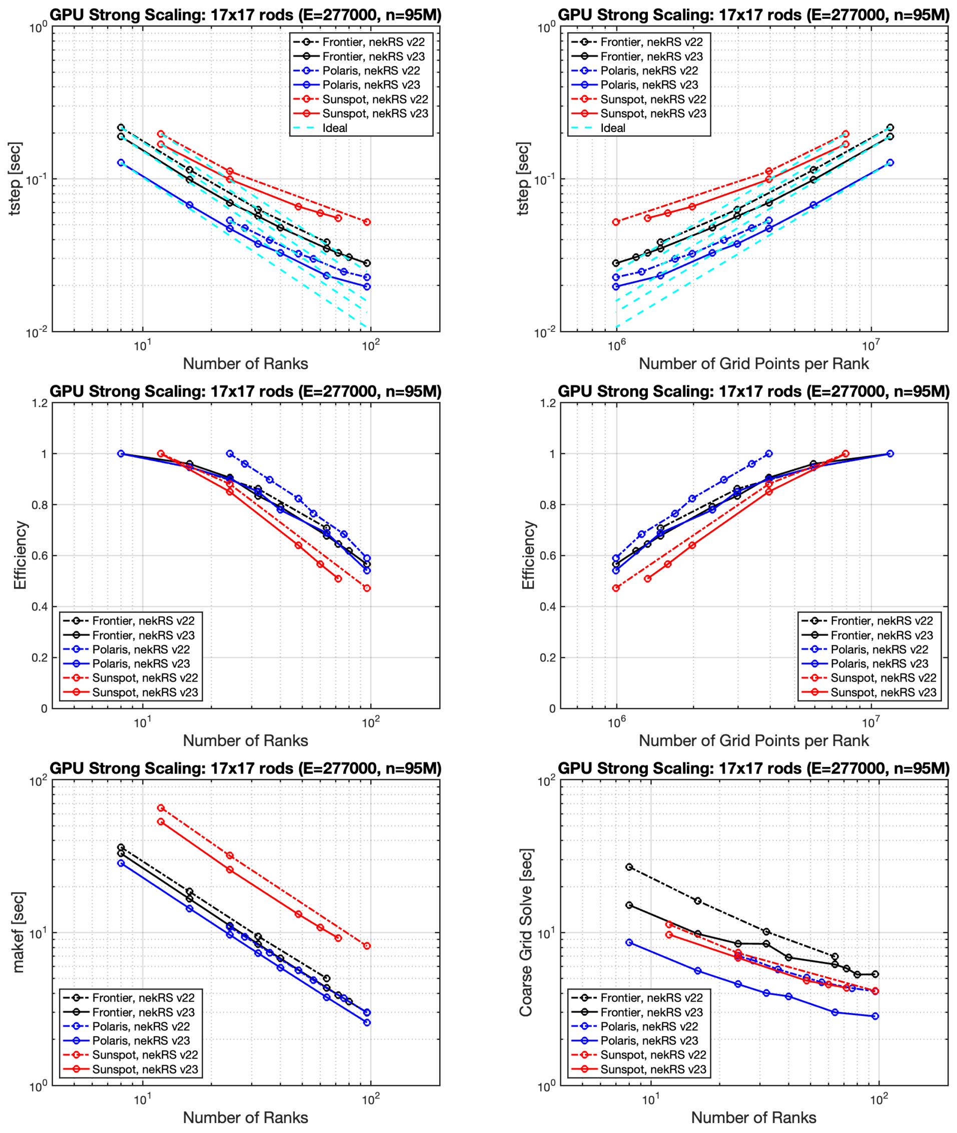


Figure 14: NekRS ExaSMR strong-scaling performance on Sunspot in comparison with Frontier and Polaris for 17 × 17 rod-bundle with 10 layers.

The NekRS team has been working in collaboration with ALCF and Intel to port NekRS to Aurora for SMR and Wind applications.

Simulations on Aurora to date have been limited to a small number of tiles. Preliminary results provided in Table 1 show that the single-PVC Aurora performance is comparable to Sunspot. Both yield an $\approx 1.45\times$ speedup when moving from one MPI rank per node to two MPI ranks per PVC (i.e., one rank per tile) for a full Navier-Stokes simulation of a single fuel rod in an ExaSMR geometry using total number of grid points of 2.4M. The single-tile runs ($n/P = 2.4$M) sustain 0.9 TFLOPS per rank, while the two-tile runs ($n/P =$ 1.2M) sustain 0.65 TFLOPS per rank, or 1.3 TFLOPS per PVC.

Further insight to initial results on Intel are provided by Figs. 14–15, which show strong-scaling results for 17×17 ExaSMR rod-bundle studies at two different resolutions (95M and 161M points, respectively), using NekRS V22 and V23. For each case, we run two ranks per Intel PVC on Sunspot (one tile), two ranks per AMD MI250X on Frontier (one per GCD), and one rank per NVIDIA A100 on Polaris.

The smaller case of Fig. 14 shows that NekRS V23 is slightly faster than V22, but this discrepancy all but disappears for the larger case of Fig. 15. Otherwise, the performance variation with problem size

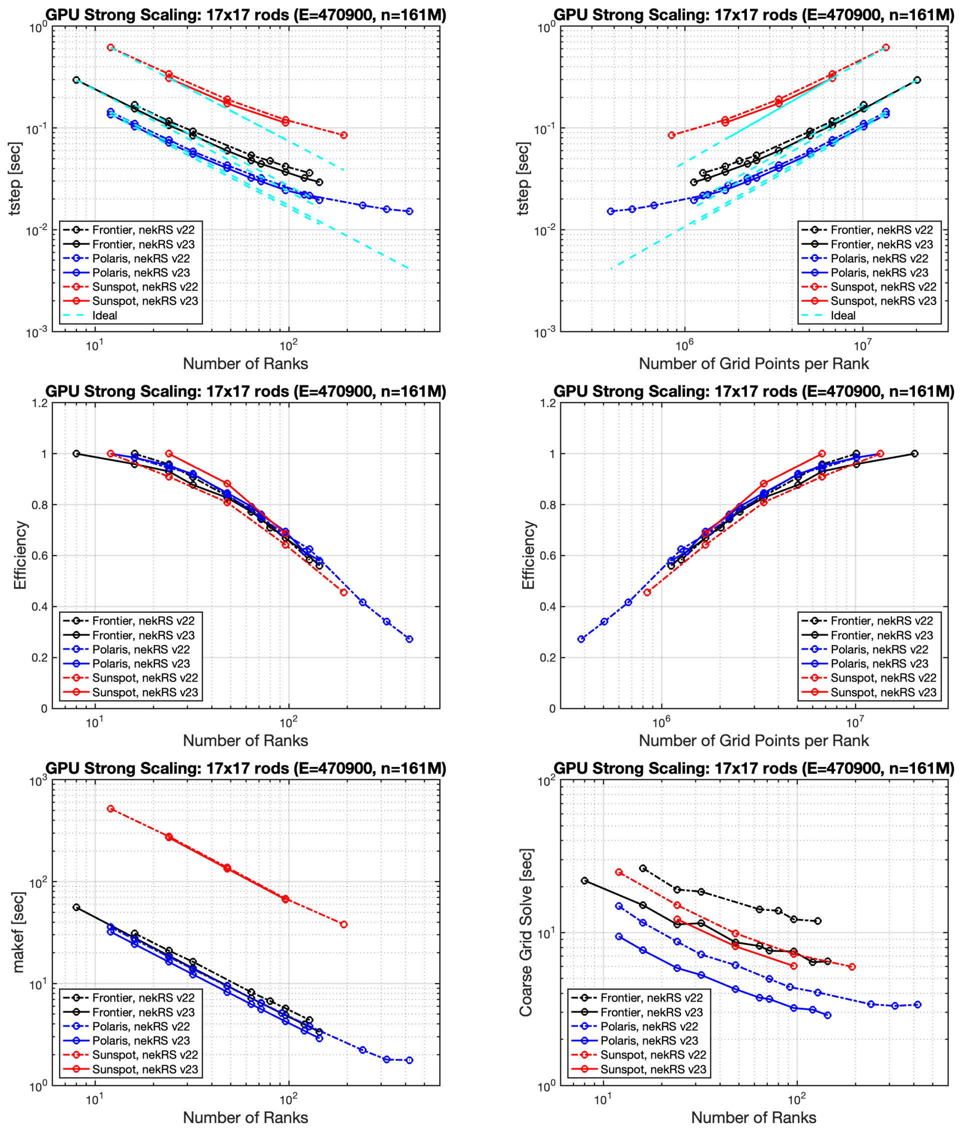
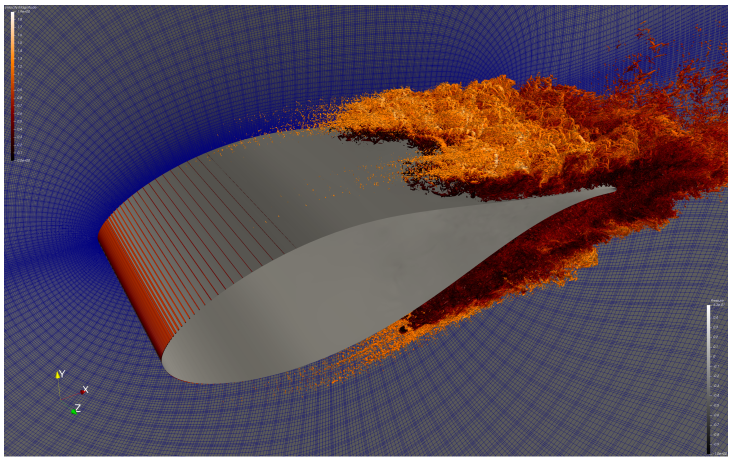


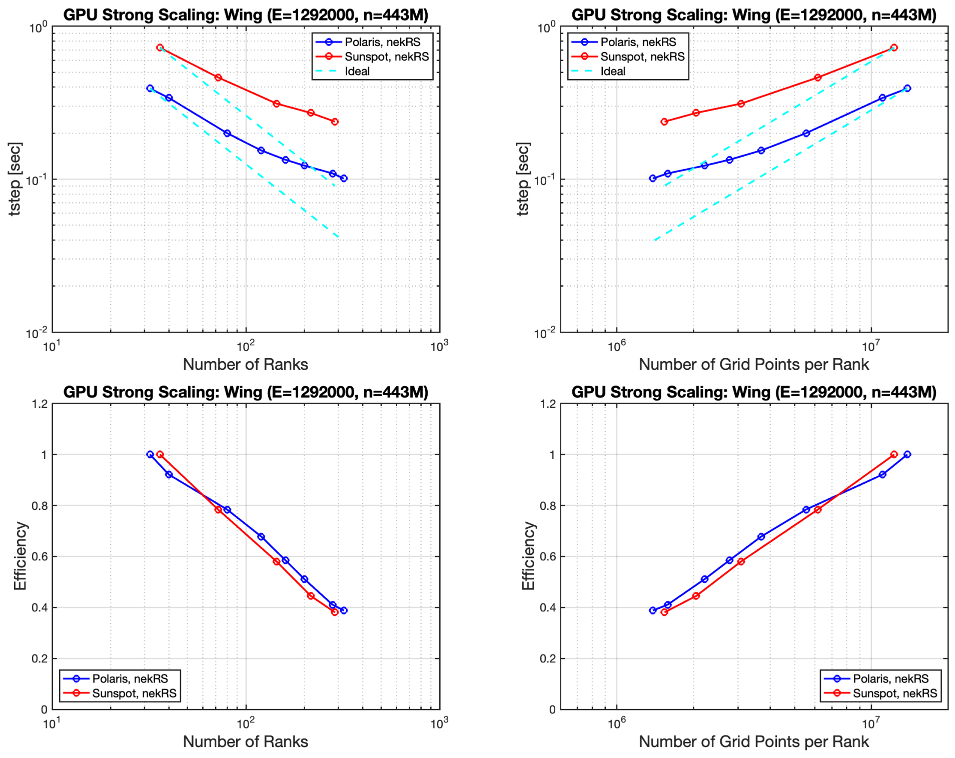
Figure 15: NekRS strong-scaling performance on Sunspot in comparison with Frontier and Polaris for 17 × 17 rod-bundle with 17 layers.

(95M vs. 161M) and version (V22 vs. V23) is not significant, with 80% parallel efficiency being realized for $n/P = n_{0.8} \approx$ 2M–3.5M across all cases.

3.2 NekRS Performance on Sunspot for Wind Applications

Figure 16 demonstrates preliminary performance results on Sunspot using $E = 1,292,000$, $N = 7$, and $n = 443$M. The top figure in Fig. 16 shows simulation of a wall resolved LES (WRLES) of flow over an AH93-W-257 airfoil at $Re_c = 100,000$ and the angle of attack, $AoA = 0$. Using a restart file from the WRLES simulation, we performed 1000 step runs on Sunspot and Polaris with Δt =3.25e-5 using BDF2 for the timestepping and demonstrated the scaling results in Fig. 16. With the non-optimized advection kernels of NekRS on Sunspot, the performance is 2X slower than Polaris. Table 2 demonstrates t_{step} measured by the average time per step using 100 steps from 901 to 1000 steps. For these simulations, the iteration numbers for velocity (v_i) and pressure (p_i) are relatively high with 2 and 7, respectively, and thus t_{step} is more than 4.6e-1 s on Sunspot and 1.9e-1 s on Polaris at the parallel efficiency of 80% using more than $n/P = 6$M on Sunspot and Polaris.





GPU Strong Scaling: Wing (E=1292000, n=443M)
tstep [sec]
Efficiency
Number of Ranks
Number of Grid Points per Rank
Polaris, nekRS
Sunspot, nekRS
Ideal

Figure 16: NekRS simulation of a wall resolved LES (WRLES) of flow over an AH93-W-257 airfoil with $Re_c = 100,000$ (top). NekRS strong-scaling performance on Sunspot in comparison with Polaris for turbine-blade simulations (bottom).

NekRS Strong-Scaling for AH93-W-257 airfoil						
NekRS on Sunspot						
node	tile	n/tile	v_i	p_i	$t_{step}(s)$	P_{eff}
3	36	1.2310e+07	1.94	6.77	7.2467e-01	100
6	72	6.1549e+06	1.96	6.92	4.6239e-01	78.36
12	144	3.0775e+06	1.97	6.73	3.1224e-01	58.02
18	216	2.0516e+06	1.95	6.76	2.7138e-01	44.50
24	288	1.5387e+06	1.98	6.92	2.3794e-01	38.07
NekRS on Polaris						
node	GPU	n/GPU	v_i	p_i	$t_{step}(s)$	P_{eff}
4	32	1.3849e+07	1.97	6.92	3.9200e-01	100
5	40	1.1079e+07	1.99	7.08	3.4031e-01	92.1
10	80	5.5394e+06	1.98	6.80	1.9999e-01	78.4
15	120	3.6930e+06	1.95	6.86	1.5414e-01	67.8
20	160	2.7697e+06	1.95	6.61	1.3406e-01	58.4
25	200	2.2158e+06	1.97	6.79	1.2267e-01	51.1
35	280	1.5827e+06	1.95	6.80	1.0910e-01	41.0
40	320	1.3849e+06	1.98	6.83	1.0106e-01	38.7

Table 2: NekRS strong-scaling on Sunspot for airfoil simulations, shown in Fig. 16 (top), in comparison with Polaris. $E = 1,292,000$, $N = 7$, and $n = 443$M.

3.3 SYCL Backend in MFEM

The MFEM SYCL single source programming kernels relies on the `SYCL 2020` specification, which is based on `C++17`. The following capabilities are incorporated into this new backend:

- Unified Shared Memory (USM), which is required for pointer-based programs to function without SYCL `buffers` and `accessors`. This is necessary to ensure that the MFEM code, which specifically describes the inputs and outputs of each kernel, does not need to be modified.
- Support for MFEM's internal threading (`MFEM_FOREACH_THREAD`) via low-level compiler `intrinsics`. Without modifying the code, MFEM SYCL can now target NVIDIA, AMD, and Intel backends.
- New ways to declare shared memory variables inside MFEM's kernels through either:
  - Static shared memory: which requires Intel's `sycl_ext_oneapi_work_group_local` compiler extension which defines a SYCL class template inspired by the C++ thread_local and the CUDA `__shared__` keywords, or:
  - Dynamic shared memory: conform to the SYCL specification of declaring shared memory variables, which also maps well to NVIDIA's and AMD's possibilities, but requires code modifications.

  Relying on a vendor extension prevents the backend from being portable; hence, this method is not actually ideal for supporting a SYCL backend that conforms to the standard. The use of dynamic shared memory is new to MFEM and requires some adjustments. For example, the programmer needs to declare explicitly the amount of shared memory that is required by each thread block and make use of a new object in order to access this memory. This new way of declaring variables was not accessible through MFEM because it could have led to performance losses. However, while developing this new feature, the team was able to match the performances of the static kernels on the GPU with these dynamic shared variables without using ahead-of-time instantiated kernels or through the use of just-in-time compilation. However, such compilation strategies are still needed to match the CPU performances from a single source kernel. The next major release of MFEM will be the goal for the necessary changes to incorporate the SYCL backend while keeping the kernel's performance intact.

The MFEM SYCL backend has been tested with Intel's DPC++ compiler which uses LLVM/Clang, and with the OpenSYCL compiler which can target OpenMP, CUDA and HIP/ROCm. MFEM's SYCL backend implementation has been verified on CPU (hosts) and on Intel CPU (OpenCL) and GPU hardware.

3.4 Ongoing MAGMA SYCL Porting Work

The UTK team has continued efforts to port MAGMA for Intel GPUs using SYCL. Recent activities include ongoing investigations of issues in the dense linear algebra routines that make up the core of MAGMA, as well as porting sparse linear algebra functionality. Both fronts have presented challenges. For the dense routines, we have encountered odd bugs in some versions of Intel software. For example, MAGMA is currently making significant use of Intel's `dpct` headers, which provide SYCL implementations of many "CUDA-like" features that are not part of the SYCL standard. The Intel conversion tool initially recommended `dpct`'s matrix memory copy routine (`dpct::matrix_mem_copy`) to replace the matrix memory copy routines from cuBLAS (`cublas[Get/Set]MatrixAsync`, `cudaMemcpy2DAsync`), but we have experienced test failures at specific matrix sizes when using this routine. Replacing the code with the latest version of the Intel 2D memcopy extension (`ext_oneapi_memcpy2d`), however, fixes the test failures, so we will transition to use this extension in the official SYCL branch of MAGMA. With the help of Thomas Applencourt and Colleen Bertoni at ALCF, we have also found a bug in MKL ZGEMM, which behaves differently depending on whether one uses implicit scaling, with both tiles of the GPU, or explicit scaling, with one tile. This has been reported to Intel.

For the sparse porting work, a challenge thus far has been MKL's lack of complex type support for sparse linear algebra routines on GPUs, though this is expected to change soon. MKL sparse also only supports one sparse matrix representation, CSR, while the CUDA MAGMA code (and cuSPARSE) supports other types, like CSC, COO, and ELL. Thus, the initial release of the sparse functionality for MAGMA's SYCL backend will not provide full support compared to the CUDA or HIP versions, though we will continue to add interfaces as MKL sparse support expands. Our current strategy of maintaining separate code for SYCL BLAS routines (as compared to HIP, which is autogenerated from CUDA), which does result in a heavier code maintenance burden, allows us more freedom to make changes in order to add our own custom support of some sparse linear algebra features not covered by MKL.

libCEED MAGMA backend for non-tensor basis functions on PVC. In the NDA appendix of the previous CEED milestone report, we presented some initial results for the libCEED MAGMA backend non-tensor basis performance on PVC, including direct comparisons with A100 and MI250X. The MAGMA backend of libCEED has two separate modes: for lower-order basis functions–defined by the backend as having both total number of basis points and total number of quadrature points per element less than 40–there are specialized batch GEMM kernels which have been shown to achieve better performance than standard GEMM/batch GEMM available in the MAGMA library or vendor libraries. These kernels use the runtime compilation framework of the other libCEED CUDA/HIP backends. For higher-order basis functions, the MAGMA backend employs a "GEMM selector" which uses lookup tables of GEMM tuning data to select a standard GEMM or batch GEMM configuration based on the size of the problem.

An initial experimental libCEED MAGMA SYCL backend using the standard GEMM selector has been created. However, our current priority is improving the performance of the specialized GEMM kernels for lower-order basis functions. Performance optimization on Intel Data Center GPU Max Series (code name PVC) is ongoing for this workload. In the following, we present a performance case study focused on kernels where (P, Q) = (20, 11), with P as the number of basis function nodes per element and Q as the number of quadrature points per element. This corresponds to standard third-order Lagrange tetrahedron elements in MFEM with a constant-coefficient `DiffusionIntegrator`. For simplicity, we also use this number of Q points for the interpolation tests, as this is what the current MAGMA backend uses in its lookup tables, but future work should also consider higher numbers of quadrature points, as would generally be used for the mass operator on tetrahedrons. For all experiments, each basis action application was repeated 11 times, with the first application treated as a warm-up and the following 10 averaged to produce the final timings.

Comparison of oneAPI versions. The previous results shown in the prior CEED milestone report were obtained with the "oneapi-prgenv/2022.10.15.006.001" module. We first compare performance with the current (as of this writing) default module on Sunspot, "eng-compiler/2023.05.15.006," to see if there are any significant changes. In Fig. 17, performance is shown versus N, which is equal to the number of basis components times the number of total elements in the mesh. For each N, a particular batch size tuning parameter, NB, has been selected in order to determine the performance; this will be discussed more below. The NB value for a given N may not be the same for each oneAPI version. We find that the difference between the two oneAPI versions is minor, except for the transpose mode gradient kernel, where the older results are significantly better for larger N. The newer module has immediate command lists turned on by

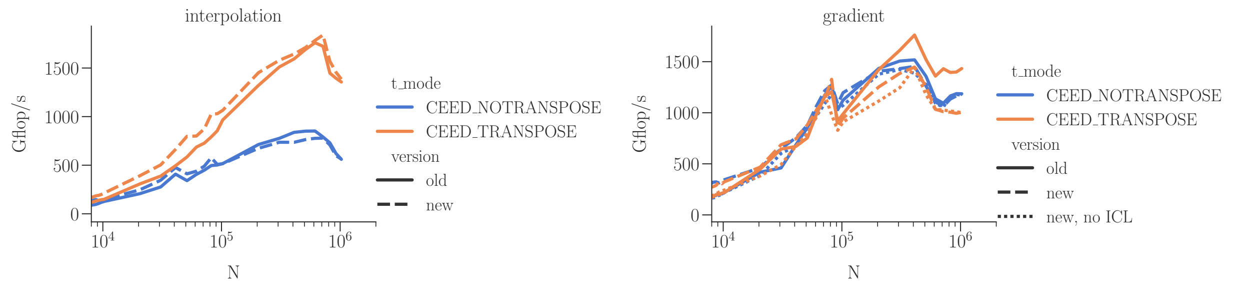


Figure 17: Comparing the performance of older (oneapi-prgenv/2022.10.15.006.001) and newer (eng-compiler/2023.05.15.006) oneAPI distributions on one tile of PVC. For the gradient kernel, we also consider the newer distribution with immediate command lists turned off, as was the default setting with the older module. All cases use large GRF mode and subgroup size/SIMD width 32.

default, while the older module did not. Thus, for the gradient kernel, we also include a case with the newer module but no immediate command lists ("new, no ICL"), to determine whether this is contributing to the gap in performance. However, as turning immediate command lists off decreases the performance of the newer module, we conclude the decrease is due to some other change in the newer compiler or runtime.

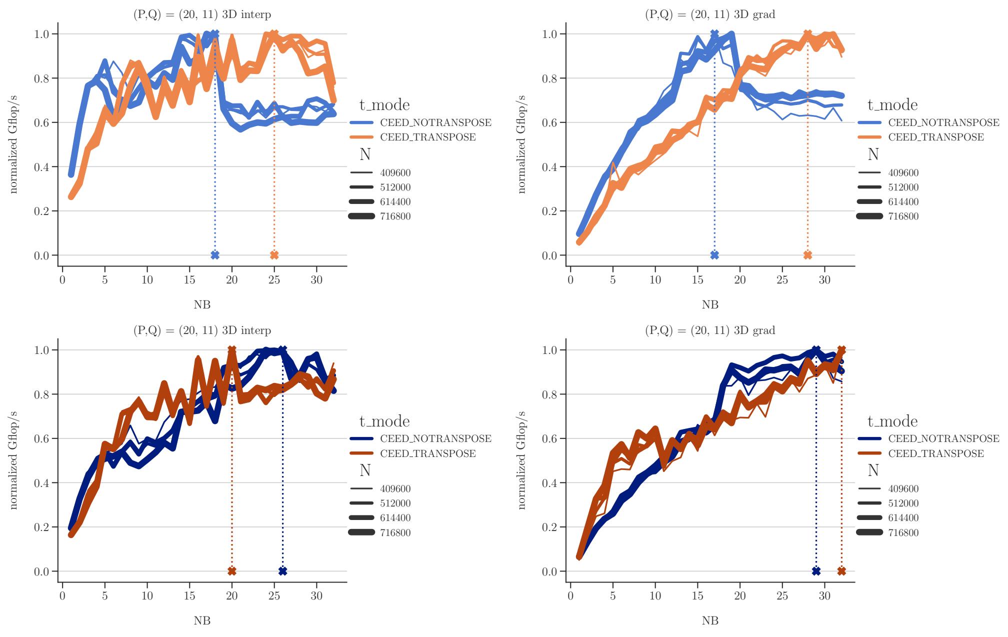


Figure 18: Selecting the best interpolation NB for a group of values of N, on one tile of PVC. The top row shows a subgroup size of 32, for interpolation and gradient basis actions, while the bottom row shows subgroup size 16. Dashed vertical lines indicate the selected best value for NB. All cases use large GRF mode.

Effects of subgroup size. We also investigated the effects of changing the subgroup size/SIMD width of the kernels. For PVC, we can choose either 32 (used in all previous results) or 16. This size is analogous to a warp for CUDA. In Figure 18, we show the effect of changing the SIMD size on the selection of the NB tuning parameter. For every value of N, NB was varied from 1 to 32, inclusive of every value in between. To find the best choices for NB, values of N were split into five groups with the same process that the libCEED MAGMA backend currently uses to select a specialized kernel to run for a particular problem size. Within each group, the performance for each N was normalized by the highest-rated GFLOP/s rate for that N, such that we can compare relative performance across the different values of NB within the group. To select the "best" NB for the group, we choose the value which maximizes the minimum performance across the group. Figure 18 illustrates this selection within one N group for the interpolation (left) and gradient kernels, with SIMD32 on the top row and SIMD16 on the bottom. The chosen NB values are indicated by the vertical lines. We clearly see the effect of changing the subgroup size, not only on the best selected value, but in the performance trends for the non-transpose kernels as NB grows.

While the effect on the tuning process is evident, the question remains whether using SIMD32 or 16 substantially changes the overall performance for a particular N. In Figure 19, we show both SIMD versions, along with the previous A100 and MI250X results. Note that in all of the following results, we are using one tile of PVC and one GCD of MI250X, but the entire A100 GPU. An interesting trend is that SIMD16 increases performance for the non-transpose application of both interpolation and gradient basis actions, yet SIMD32 does better for the transpose case. Unfortunately, both still lag the other architectures significantly, with the exception of the transpose interpolation case, where the current SYCL kernels are competitive with the HIP/MI250X performance until reaching the largest values of N that we considered.

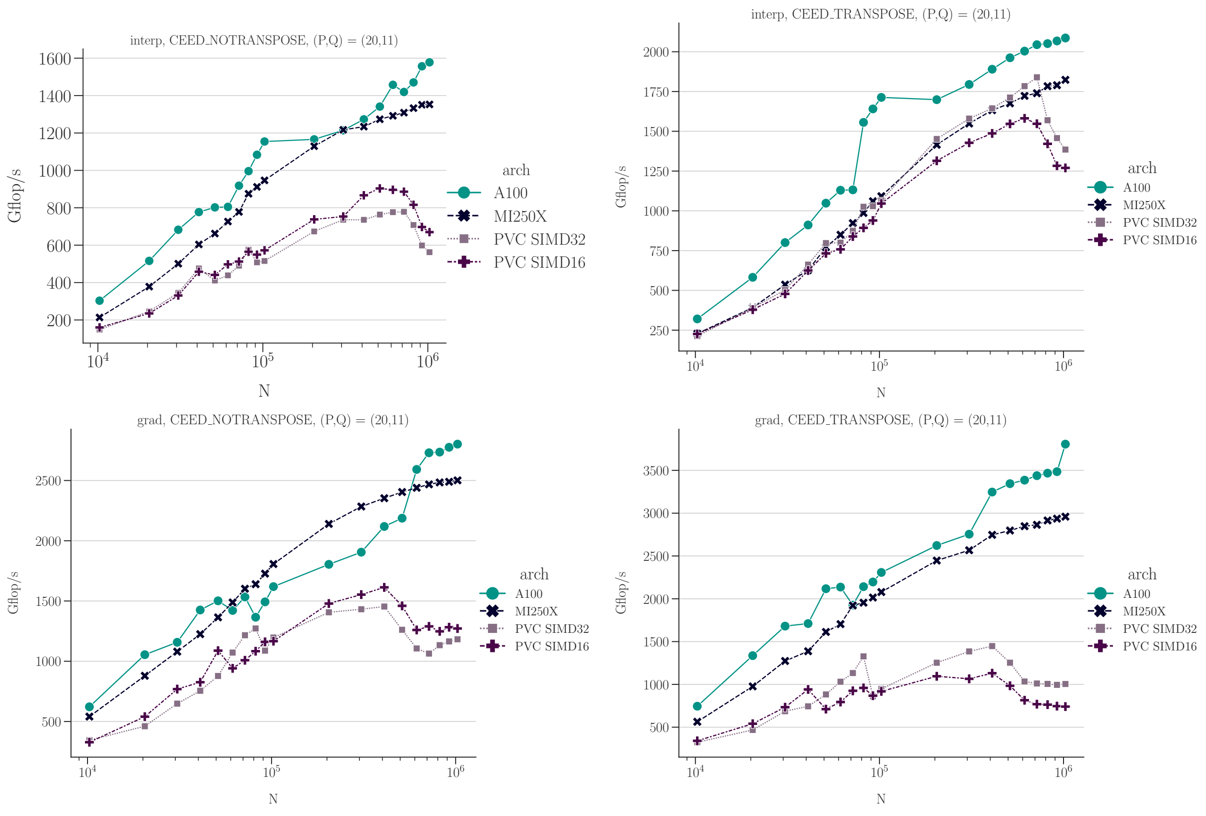


Figure 19: Comparing performance of the specialized MAGMA backend non-tensor interpolation (top row) and gradient (bottom row) kernels. Tests performed on NVIDIA A100 (CUDA 11.4), one GCD of AMD MI250X (ROCm 5.4), and one tile of Intel PVC (eng-compiler/2023.05.15.006).

Occupancy and IGC compiler output. To further investigate the performance, we examined IGC compiler output for several metrics which might be expected to affect performance of the kernels. The three highlighted values – register spill size, memory bank conflicts, and number of sync instructions – are normalized across values of NB and shown in the left plots of Figures 20–22. The normalized performance of the kernels is also shown for a small, medium, and large value of N. The right side of the figures contains occupancy information for the same values of N. Occupancy here is “total” tile-based occupancy, i.e., taking the entire tile into account, rather than just the number of workgroups that can theoretically fit on one Xe-core at a time. If there are too many groups to fit on the tile at one time, the reported occupancy is a weighted average of the occupancy of “full” sets of groups and the occupancy of a final “leftover” set of groups. That is why the occupancy is always lower for the smallest value of N – there is not enough work to fill the tile. We also note that the maximum possible occupancy is only 50%, as we are using the large register file option for these kernels (previously shown to improve performance), which cuts the total possible subgroups per Xe-core in half. Intel’s theoretical occupancy calculation is based on subgroup size, group size, total size, and shared memory usage. For a particular kernel, the group size is the same across all values of NB, but the total number of groups changes with NB. The shared memory usage also increases linearly with NB. In the occupancy plots, markers with a black center indicate that the number of groups per Xe-core is limited by shared memory for this configuration, rather than group size. For the interpolation figures (20 and 21), which have SIMD32 on the top row and SIMD16 on the bottom, we clearly see how the smaller SIMD size reduces register spilling and the number of configurations limited by shared memory usage.

A perhaps surprising result was that register spills and bank conflicts are not as clearly tied to performance as we expected. For example, in the non-transpose interpolation case with SIMD32 (Fig. 20), the best-performing NB for N = 40960 is one with a non-zero value for both spill size and bank conflicts, despite other options with similar theoretical occupancy. It’s also possible that register spills matter more when the total number of groups is smaller, as the performance drop at large NB for N = 409600 is much less than for the other values of N, except in the case of transpose interpolation with SIMD16 (Figure 21, bottom left), for which no kernels reported register spills.

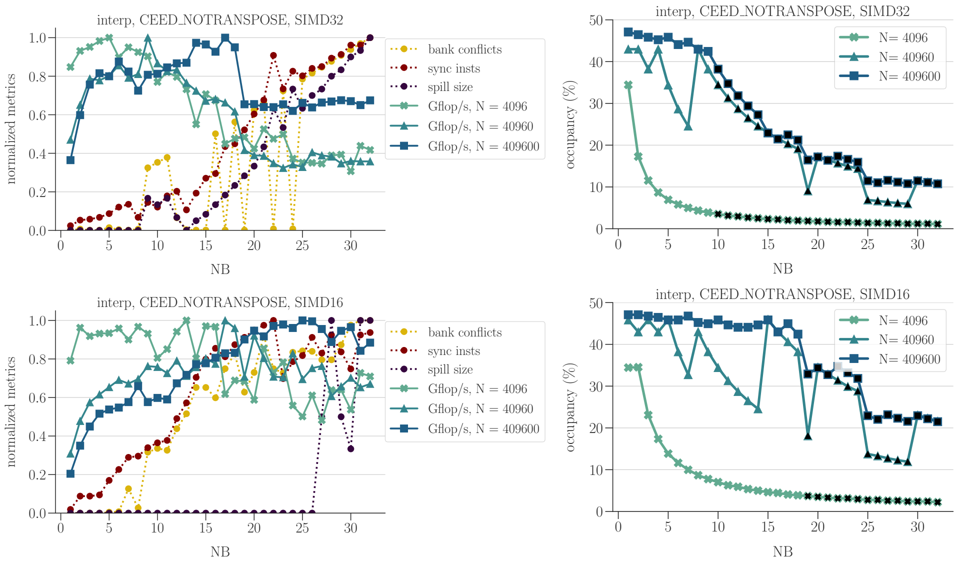


Figure 20: Non-transpose interpolation kernel occupancy and normalized (with respect to maximum among values of NB for a particular N) performance versus NB. Three values of IGC compiler output are shown for comparison, also normalized based on the maximum value. Occupancy is theoretical, based on group size, total groups, and shared memory requirements; a marker with a filled black center indicates that the occupancy is limited by shared memory usage for that configuration. The top row is for kernels compiled with SIMD width/subgroup size of 32, while bottom has size 16.

Figure 22 focuses on SIMD32 for the gradient kernels, both non-transpose (top row) and transpose (bottom row). The effect of register spills, sync instructions, and bank conflicts on performance could appear clear for the non-transpose case, yet less so for transpose, casting doubt on whether this particular set of information from the compiler can predict performance or guide development of improved kernels for SYCL.

We plan to continue our investigation with increased use of Intel performance tools.

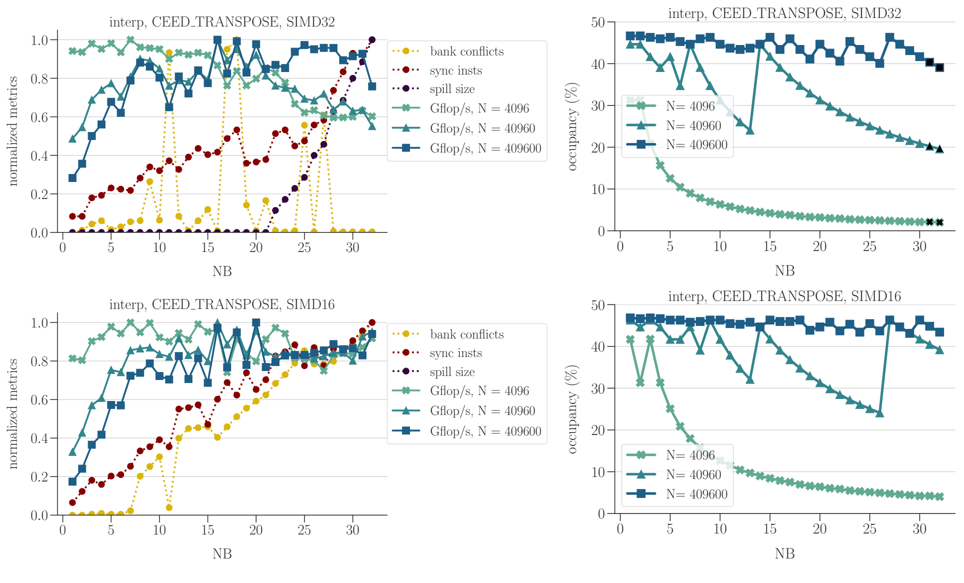


Figure 21: Transpose interpolation kernel occupancy and normalized (with respect to maximum among values of NB for a particular N) performance versus NB. Three values of IGC compiler output are shown for comparison, also normalized based on the maximum value. Occupancy is theoretical, based on group size, total groups, and shared memory requirements; a marker with a filled black center indicates that the occupancy is limited by shared memory usage for that configuration. The top row is for kernels compiled with SIMD width/subgroup size of 32, while bottom has size 16.

4. ECP APPLICATIONS SUPPORT

4.1 ExaAM

Through a collaboration with the CEED and ExaWorks teams, the ExaAM team recently developed an uncertainty quantification workflow to quantify how varying additive manufacturing (AM) build parameters affects the mechanical response of AM built parts. The collaboration with the CEED team was largely focused on accelerating the ExaConstit component which would ultimately be used in the most computationally expensive stage of the ExaAM challenge problem workflow. As reported in the past CEED-MS40 report, this collaboration was vital in optimizing a key aspect of ExaConstit which was preventing its performance on Crusher/Frontier from being faster than Summit.

ExaAM Frontier Runs. The ExaAM team gained access to Frontier at the end of March 2023. Since ExaConstit code base was already running on Crusher, it was a straightforward process to modify the Crusher build script to work with all the Frontier specific modules. We have uploaded the Frontier build script to the ExaAM uncertainty quantification workflow repo [24]. Initial testing of ExaConstit on Frontier versus Crusher revealed no performance drops, and we were still obtaining a 1.5x speed-up over the Summit runs for a 4e6 linear hexahedron test case. After the verifying performance of each component, we began stress-testing our uncertainty quantification workflow for the ExaAM challenge problem, as seen in Figure 23. More specifically, we stressed the Stage 3 Radical-EnTK [13] workflow manager component by scaling up to several hundred nodes per run as our challenge problem workflow would require 7850 high fidelity ExaConstit simulations to be run on 8000 nodes of Frontier. These 7850 simulations are the result of the n-fold Cartesian product of the varying melt pool parameters (5 variations) for Stage 1a, microstructure generation parameters (25 variations) for Stage 1b, loading conditions (21 variations) for Stage 3 and temperatures (3 variations) for Stage 3. The

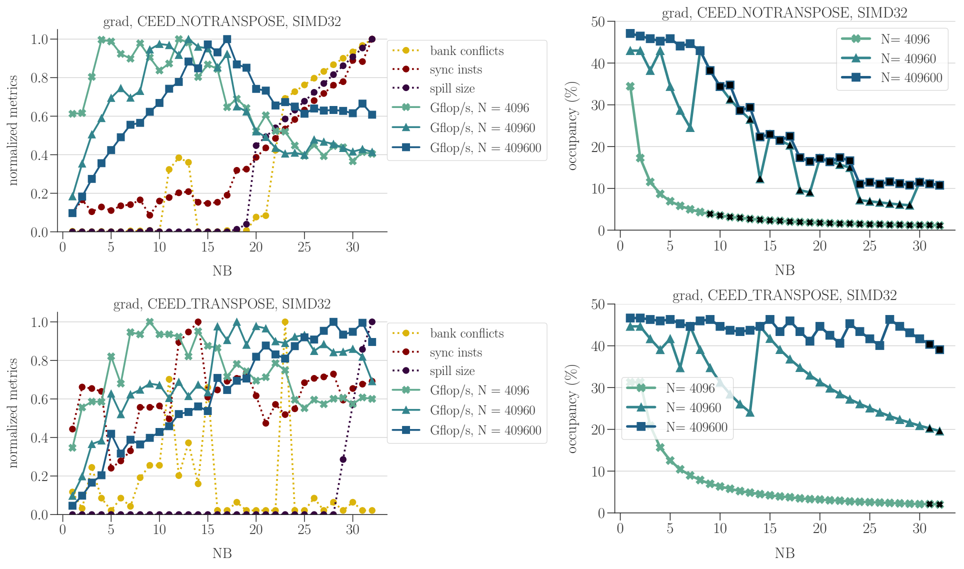


Figure 22: Gradient kernel occupancy and normalized (with respect to maximum among values of NB for a particular N) performance versus NB. Three values of IGC compiler output are shown for comparison, also normalized based on the maximum value. Occupancy is theoretical, based on group size, total groups, and shared memory requirements; a marker with a filled black center indicates that the occupancy is limited by shared memory usage for that configuration. The top row shows non-transpose kernels, while the bottom row is transpose; all kernels use subgroup size/SIMD width 32.

computational requirement for each ExaConstit simulation is the same and is 8 nodes of Frontier with 64 MPI ranks using the typical 1 CPU per 1 GPU decomposition, and each simulation requires between 10-20 minutes to complete. Therefore, even using 8000 nodes, we are only able to run 1000 concurrent ExaConstit simulations at a time. On May 14th 2023, we ran all 7850 ExaConstit runs within 2 hours and 15 minutes. Out of these 7850 simulations, only 8 failed which could be accounted to a single node failure. Our workflow manager automatically re-submitted these 8 jobs and marked their successful completion after they finished re-running. As part of the Radical-EnTK toolkit, our workflow captures the resource utilization through-out the life of the workflow run. We can see in Figure 24 that for a majority of the run we are utilizing 100% of the 8000 nodes of Frontier, and then as we end up with fewer than 1000 jobs the utilization falls below 100% as no more jobs are being submitted.

4.2 MARBL

As Lawrence Livermore National Laboratory (LLNL) will soon have in place the El Capitan exascale supercomputer, the focus of the MARBL team for the past months has been to achieve high performance on AMD-based GPUs. Reaching this task has been aided by the multiyear effort under the NNSA Advanced Simulation and Computing (ASC) program, and by the collaborations with ECP and the CEED co-design center. Previous CEED milestone reports [36, 75, 22, 59, 60] have presented the various co-design efforts between the teams, including the development of the Laghos [64, 35], Remhos [97, 5, 4], and pmesh-optimizers [38, 37] miniapps; new matrix-free algorithms for the Lagrange stage of the code [20]; GPU-based mesh optimization algorithms [23]; matrix-free remap methods [49, 50]. A comprehensive overview of that work can also be found in [119]. In this section we give a summary of the recent AMD porting efforts, including some new matrix-free numerical approaches and algorithmic developments that were required to achieve GPU performance; a more detailed technical description of the work can be found in [110]. As a result of these efforts MARBL is currently able to perform large-scale practical 3D multiphysics simulations, like the ones in Figures 25 and 26, on the latest El Capitan Early Access Systems (EAS-3) using AMD MI250X GPUs.

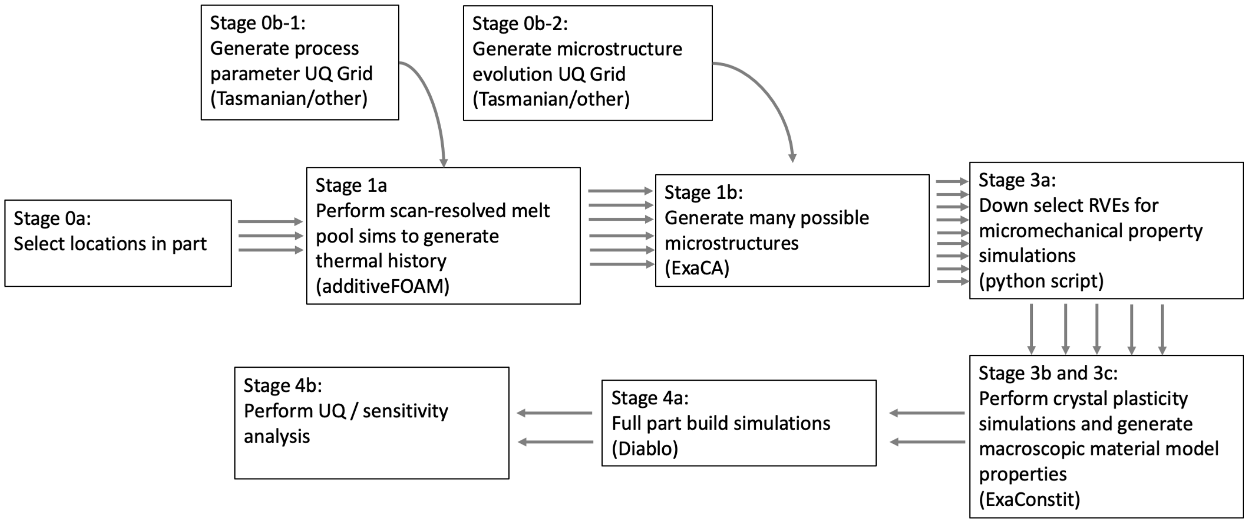

Stage 0b-1:
Generate process parameter UQ Grid (Tasmanian/other)
Stage 0b-2:
Generate microstructure evolution UQ Grid (Tasmanian/other)
Stage 0a:
Select locations in part
Stage 1a
Perform scan-resolved melt pool sims to generate thermal history (additiveFOAM)
Stage 1b:
Generate many possible microstructures (ExaCA)
Stage 3a:
Down select RVEs for micromechanical property simulations (python script)
Stage 3b and 3c:
Perform crystal plasticity simulations and generate macroscopic material model properties (ExaConstit)
Stage 4a:
Full part build simulations (Diablo)
Stage 4b:
Perform UQ / sensitivity analysis

Figure 23: A self-consistent AM UQ workflow to quantify the effects of machine parameters on AM part performance.

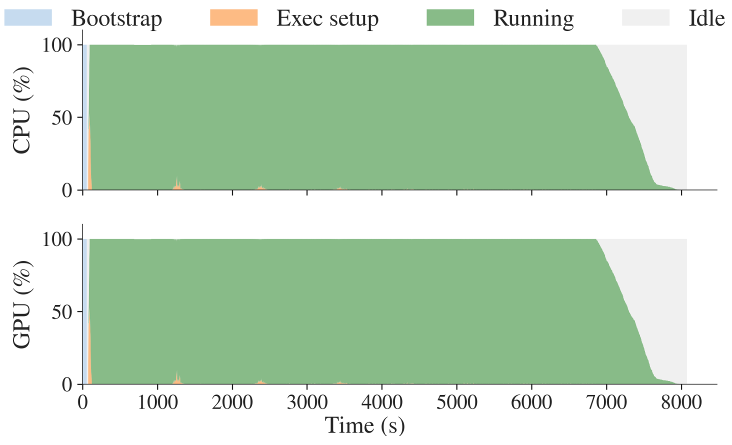


Figure 24: Resource utilization by the Radical-EnTK application (UQ Stage 3): 100% corresponds to 448,000 CPU cores (not considering 8 cores per node reserved for system processes) and 64,000 GPUs.

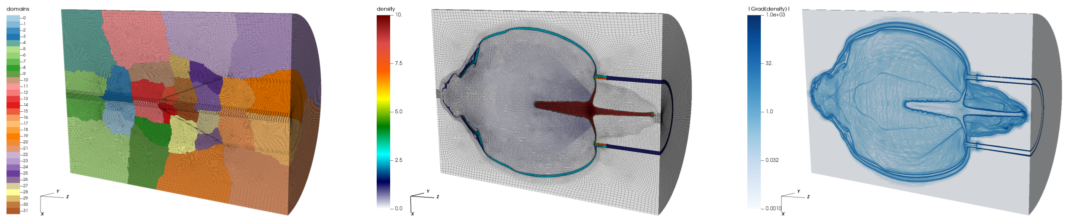


Figure 25: Results of MARBL simulation of the BRL81a shaped charge, with off-axis detonation, consisting of 114M quad points on a 3D high-order, unstructured NURBS mesh, run on 4 nodes of EAS-3 (16 MI250X GPUS, 32 MPI ranks), left to right: MPI domain decomposition across 32 ranks, mass density at final time, and density gradient (log scale) at final time.

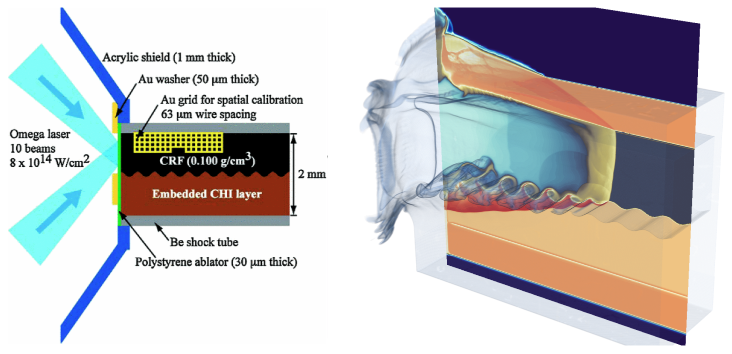


Figure 26: Schematic depiction of the laser driven Kelvin-Helmholtz Instability experiment (left) and 3D MARBL model (material density, log scale, combined with volume rendering) consisting of 600M quadrature points run on 20 nodes of EAS-3 (80 MI250X GPUs with 480 MPI ranks, 3X GPU over subscription)

AMD GPU Portability. GPU and performance development efforts in MARBL have always been made with platform portability at the forefront. MARBL leverages the RAJA Portability Suite for kernel abstractions [16], Umpire for memory allocation and pool abstractions s[17], as well as the MFEM library's memory movement abstractions [7]. Previous work and staff expertise in GPU development gained during the team's engineering efforts for ALE-Hydro on NVIDIA GPUs have greatly accelerated MARBL's AMD GPU readiness. The AMD porting work consisted of 3 main tasks: (i) compiling and linking a GPU-enabled MARBL for AMD, (ii) configuring MARBL to dispatch runtime kernels to AMD GPUs, and (iii) achieving reasonable levels of performance. The following paragraphs provide discussions about each of these tasks.

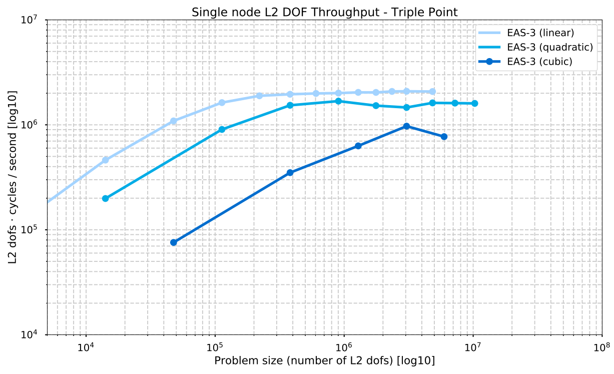
Several factors have aided MARBL's building efforts, including: expertise built up over our previous porting efforts, the BLT build system [1] ("CMake-based foundation for HPC"), and AMD's and Cray's use of LLVM-based compilers for C and C++. Most aspects of the C and C++ build port have been relatively straightforward with the exception of *relocatable device code* (RDC), which is needed when calling a GPU function from a kernel in a separate compilation unit. To simplify the build, the MARBL team has chosen to remove RDC from internal libraries that are directly controlled, by moving device function definitions to headers in the relatively few instances where RDC was (previously) required. MARBL uses templates sparingly and relies on the mostly robust and portable C++14 standard. This strategy has simplified the AMD GPU build relative to more heavily templated applications and/or those that use more modern C++ standards, especially in the early days of the EAS software stacks.

Once MARBL was built with AMD GPU support, runtime capabilities were added to the source by extending preprocessor macros to include `HIP` along with `CUDA` and by adding RAJA support for HIP to MARBL's `forall` routines. The MARBL team leveraged RAJA's support for HIP via a series of small updates to generalize CUDA-specific types, for example, the previously used `RAJA::cuda_block_x_direct` was replaced by `device_block_x_direct`. Depending on the target architecture, the latter is defined as `RAJA::cuda_block_x_direct` or `RAJA::hip_block_x_direct`. Since the RAJA changes were confined to a few common files, most of the source code did change at all. Umpire support for HIP required no MARBL source changes as device allocators were abstracted behind a `"DEVICE"` identifier that hid CUDA and HIP specific routines. MFEM also required no MARBL source changes except for a switch to `raja-hip` from `raja-cuda` as the compute policy in the input files.

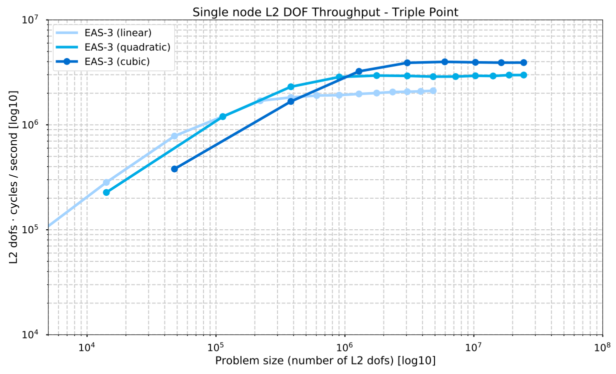
The kernel performance has been improved relative to Sierra by rewriting kernels to use more shared memory, increasing the thread block/work group size of some existing shared memory kernels, and reducing the amount of shared memory used in some existing shared memory. Further performance improvements have been achieved by following and profiling the latest compiler runtime updates, improvements in the API behavior consistency, and careful utilization of atomics and reducers for cases when multiple threads operate on the same memory in a non-read-only way. Further technical details can be found in [110].

New Multiphysics Algorithms. In parallel with performance portability developments, the MARBL team has been working on extending performant GPU capabilities beyond the ALE Hydrodynamics discussed in [119]. As an interdisciplinary team of physicists, mathematicians, and computer scientists, the CEED and MARBL teams developed and collaborated on new methods and algorithms that are designed with high-order scaling on GPUs in mind.

Matrix-free remap with interface sharpening. The MARBL and CEED teams developed a novel, fully matrix-free remap algorithm. The motivation of this works was two-fold. First, the teams had to address the degrading ALE remap throughput performance observed in [119] and, second, the previously employed remap framework can exhibit excess material diffusion in certain problems. The new algorithm is based on decoupling the material velocity from the mesh motion during remap, in order to minimize material propagation. Comparison between the new method and the existing one in terms of throughput and material diffusion are given in Figures 27 and 28.



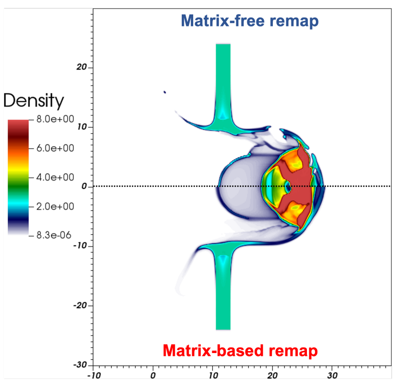
(a) Matrix-based ALE remap



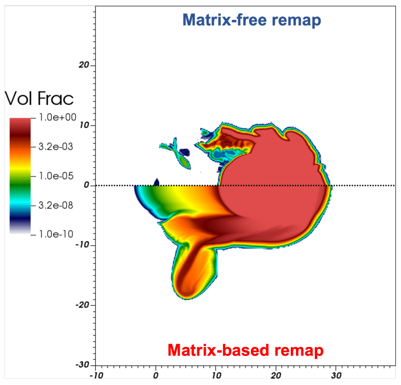
(b) Matrix-free ALE remap

Figure 27: Performance improvements achieved by switching to a matrix-free ALE remap framework on EAS-3 (AMD MI250Xs).

Saddle point linear diffusion solver. The diffusion approximation for the transport of radiation energy requires solving a sparse, global linear system which represents a diffusion operator acting over the entire computational domain. MARBL uses the so-called H(div) finite element formulation for discretizing the radiation energy diffusion equation, which requires a special type of preconditioner for efficiently solving the linear system in a scalable manner [61]. Following the work presented in [90], the MARBL team implemented a novel matrix-free, GPU-accelerated linear solver using the saddle-point formulation of the radiation diffusion equations. This solver uses the interpolation-histopolation basis from [90] to form a matrix-free preconditioner for a system that can be solved using MINRES in a number of iterations independent of the polynomial degree or the size of the problem. Numerical results demonstrate a naïve speedup of 4.5x on EAS-3 compared to CTS-1, although the matrix-based hybrid method continues to outperform on CPU systems.



(a) Density comparison



(b) Volume fraction comparison

Figure 28: The new matrix-free remap framework can sharpen material interfaces and reduce material diffusion during the remap procedure. Top simulation is the result from the matrix-free remap with interface sharpening framework. Bottom simulation is the result from the matrix-based remap framework.

Table 3: Per node floating point (64bit) and memory bandwidth performance

		CTS-1 2 18-core Intel Broadwell	**CTS-2** 2 56-core Intel Sapphire Rapids	**ATS-2** 4 NVIDIA V100s	**EAS-3** 4 AMD MI250Xs
FLOP Rate (LINPACK)	Perf	1.09 TF/s	7.8 TF/s (theoretical)	21.91 TF/s	141.78 TF/s
	Rel (CTS-1)	1.00×	7.16x (theoretical)	20.01×	130.07×
	Rel (ATS-2)	-	-	1.00×	6.47×
Memory Bandwidth (STREAM)	Perf	136 GB/s	460 GB/s	850 GB/s × 4 3.4 GB/s	1.35 TB/s × 8 10.08 TB/s
	Rel (CTS-1)	1.00×	3.3×	25.0×	74.12×
	Rel (ATS-2)	-	-	1.0×	2.9×

Throughput Scaling Study. This section presents throughput scaling comparisons between EAS-3, ATS-2 (Advanced Technology System 2, Sierra), and two CPU architectures, CTS-1 and CTS-2 (Commodity Technology Systems at LLNL), described in Table 3. The corresponding throughput results on the Triple Point benchmark are shown in Figure 29. For each architecture, the problem size is increased until the node runs out of memory; as some architectures have more memory, some curves have more points. It is evident that the CPU-based CTS-1 and CTS-2 saturate almost immediately, while the GPU-based ATS-2 and EAS-3 require millions of DoFs before reaching a steady state. The EAS-3's large memory advantage allows it to scale nearly an order of magnitude further than the existing GPU architecture, ATS-2.

4.3 ExaSMR

This section discusses our collaborative efforts between ECP's ExaSMR (Exascale Small Modular Reactor) and Co-design Center for Efficient Exascale Discretizations (CEED) teams. We present simulation capabilities for ExaSMR geometries and performance analysis for neutronics and thermal hydraulics on pre-exascale and exascale systems using ExaSMR's ENRICO platform consisting of OpenMC, Shift and NekRS.

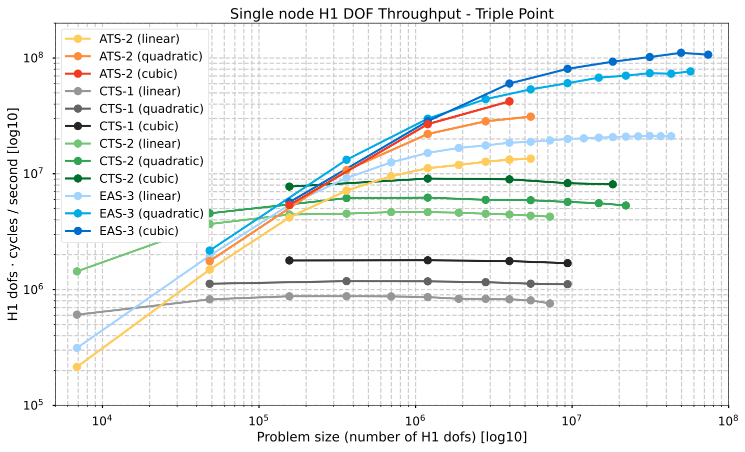


Figure 29: Triple Point linear, quadratic, and cubic throughput

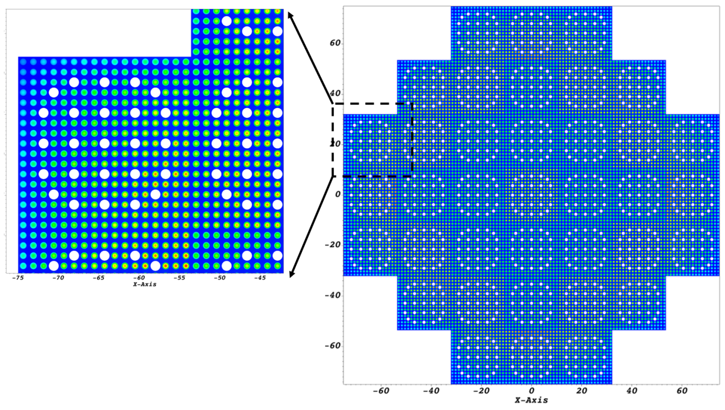


Figure 30: Radial temperature distribution ($z = 100$ cm) from coupled OpenMC–NekRS simulations of the full-core model.

Coupled nekRS-OpenMC simulations of SMR full core. After the successful verification of the NekRS-OpenMC coupling capability, a comprehensive simulation of an SMR full core was conducted on the Summit supercomputer at OLCF. Each simulation job utilized 300 Summit nodes, with a polynomial order of $N = 3$ for NekRS. Notably, the majority of the simulation time was dedicated to the NekRS solver, with OpenMC running for only 54 minutes out of the total 360-minute runtime, representing 15% of the overall computation.

Figure 30 presents the radial temperature distribution across the entire SMR reactor core at the axial mid-plane (z=100 cm). The plot reveals distinct radial rings within each fuel pin, depicting the transitions from the fuel region to the gap and cladding. In this comprehensive simulation, a total power of 160 MW is deposited in the full core model, resulting in a temperature rise of 56 K (100.8 F) in the coolant flow, which closely aligns with the design specifications of NuScale (ΔT=100 F). Notably, the peak fuel and coolant temperatures observed are 1212.0 K and 587.7 K, respectively. It is important to note that the turbulence modeling options employed in this study are intentionally simplistic. As part of ongoing efforts to enhance the simulation accuracy, additional simulations are planned with more sophisticated turbulence modeling and higher resolution to evaluate and quantify the uncertainty associated with the current results.

Hybrid RANS/LES modeling. The RANS/LES hybrid approach is a novel and promising strategy that has been introduced to enhance the simulation of a nuclear reactor core. In this approach, the core is divided into different regions or zones, with a portion of the core simulated using Large Eddy Simulation (LES) while

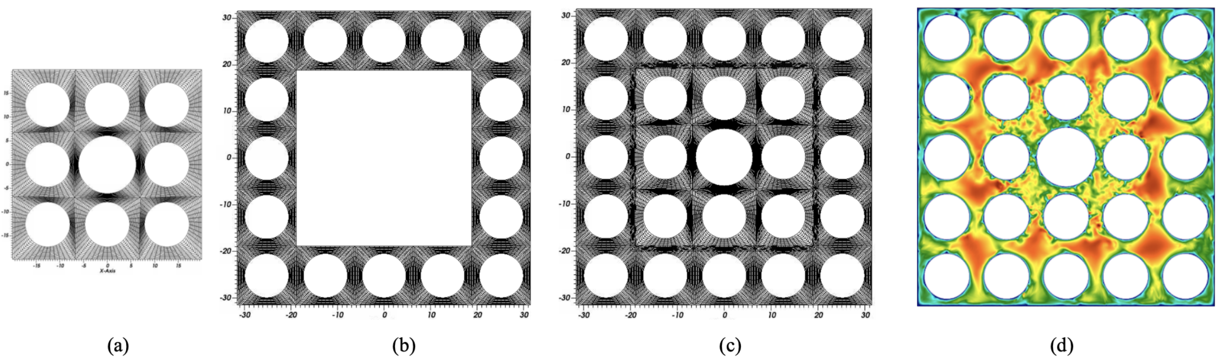


Figure 31: Multizonal mesh cross-sections of a 5×5 bundle: (a) 3×3 subchannel, (b) 5×5 subchannel, (c) Overlapped subchannels, (d) Demonstration of zonal hybrid approach for a rod bundle case Instantaneous velocity magnitude on a streamwise cross section.

the remaining regions are handled by Reynolds-Averaged Navier-Stokes (RANS) modeling. This zonal hybrid strategy aims to combine the strengths of both approaches, leveraging the accuracy and capability of LES in resolving turbulent flow features in specific regions, while still benefiting from the computational efficiency of RANS for the majority of the core. The preliminary results obtained for a subset of the reactor using this approach have shown great promise, motivating further investigation and development to ultimately improve the full core simulation without significantly increasing computational overheads. The project has pursued two parallel methods: the first involves modeling the hybrid RANS-LES using a multizonal mesh, and the second incorporates RANS and LES coupling through overlapping meshes, utilizing the overset mesh (a.k.a. "neknek") feature to achieve this ambitious goal. As an illustrative example, a relatively simple 5×5 rod bundle is considered, and the computational grids for both the multizonal and overlapping mesh approaches are depicted in Fig. 31 (a)–(c). These grid representations highlight the flexibility and versatility of the hybrid approach, showcasing the potential to effectively optimize computational resources while obtaining high-fidelity simulation results for complex reactor geometries. A demonstration of the capability is shown in Fig. 31 (d) demonstrating a solution with RANS and LES characteristics.

ExaSMR Frontier runs. We demonstrated NekRS performance results for the full-core ExaSMR geometries on full Frontier in earlier Section 2.1. Some earlier studies on Frontier for the ExaSMR's 17 × 17 rod-bundle geometries with different problem sizes were demonstrated in comparison to Crusher, Spock, Polaris, Perlmutter, ThetaGPU and Summit in [77].

4.4 ExaWind

Through a collaboration between the ECP's ExaWind project at the National Renewable Energy Laboratory (NREL) and the Center for Efficient Exascale Discretizations (CEED), the Nek5000/RS team has recently developed wall-modeled large-eddy simulation (WLES) approaches for atmospheric boundary layers (ABLs) [74, 78]. Such a capability is critical for applications such as wind farm analysis and dispersive transport in urban canyons. Here, we discuss the governing equations for WLES in Nek5000/RS along with new subgrid-scale (SGS) turbulence modeling approaches [78, 115]. We also present convergence results, demonstrating that NekRS provides high-order convergence.

We also note that the modeling approaches we developed have been applied to the turbulent modeling studies [62] that are in collaboration with Dr. Lehmkuhl's group at Barcelona Computing Center (BSC) with an Argonne postdoc, Vishal Kumar.

ExaWind Subgrid-Scale Modeling. To quantify the effects of numerical modeling and discretization choices, the ABL community has set up a sequence of benchmark problems, including the GEWEX (Global Energy and Water Cycle Experiment) Atmospheric Boundary Layer Study (GABLS) [15]. These benchmarks represent the atmospheric boundary layer in regional and large-scale atmospheric models and are considered important for improved modeling in the study of wind-energy, climate, and weather on all scales [98].

For the atmospheric LES, we solve the incompressible Navier–Stokes (NS) and potential temperature equations in a *spatially filtered* resolved-scale formulation. The governing equations are expressed in nondimensional form as

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\bar{\rho}} \frac{\partial \bar{p}}{\partial x_i} - \frac{\partial \tau_{ij}}{\partial x_j} + f_i \tag{1}$$

$$\frac{\partial \bar{u}_j}{\partial x_j} = 0, \tag{2}$$

$$\frac{\partial \bar{\theta}}{\partial t} + \bar{u}_j \frac{\partial \bar{\theta}}{\partial x_j} = -\frac{\partial \tau_{\theta j}}{\partial x_j}, \tag{3}$$

where $\bar{u}_i$ is the ith component of the resolved-scale velocity vector, $\bar{\rho}$ is the density, $\bar{p}$ is the pressure, and $\bar{\theta}$ is the potential temperature in the resolved scale. Here, f_i accounts for the Coriolis acceleration and buoyancy force. In the case of the high-pass filter model discussed below, f_i would also include a damping term of the form $-\chi HPF(u_i)$, where χ is an order-unity multiplier and HPF is a spatial filter function that allows grid-scale fluctuations of the argument to pass through so that their amplitude is damped in time. In addition, we have the stress tensors in the momentum and energy equations, τ_{ij} and $\tau_{\theta j}$, respectively, involving SGS turbulent modeling terms:

$$\tau_{ij} = -\frac{2}{Re} S_{ij} + \tau_{ij}^{sgs} = -\frac{1}{Re}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) + \tau_{ij}^{sgs}, \tag{4}$$

$$\tau_{\theta j} = -\frac{1}{Pe} \frac{\partial \bar{\theta}}{\partial x_j} + \tau_{\theta j}^{sgs}, \tag{5}$$

where Re is the Reynolds number, Pe is the Peclet number, S_{ij} is the resolved-scale strain-rate tensor, and τ_{ij}^{sgs} and $\tau_{\theta j}^{sgs}$ are the SGS stress tensors that require modeling.

We consider the GABLS benchmark [14], illustrated in Figure 32, which is a well-documented stably-stratified flow problem, having the ground temperature being cooler than the air temperature with continued cooling over the simulation duration. The computational domain is defined on $\Omega = L_x \times L_y \times L_z = 400$ m $\times$ 400 m $\times$ 400 m with the streamwise direction in x, the spanwise direction in y, and the vertical direction in z. We define an initial condition at $t = 0$ with constant velocity in the streamwise direction equal to geostrophic wind speed of $U = 8$ m/s. The initial potential temperature is 265 K in $0 \le z \le 100$ m and linearly increased at a rate of 0.01 K/m in 100 m $\le z \le 400$ m, with the reference potential temperature of 263.5 K. An initial perturbation is added to the temperature with an amplitude of 0.1 K on the potential temperature field for $0 \le z \le 50$ m.

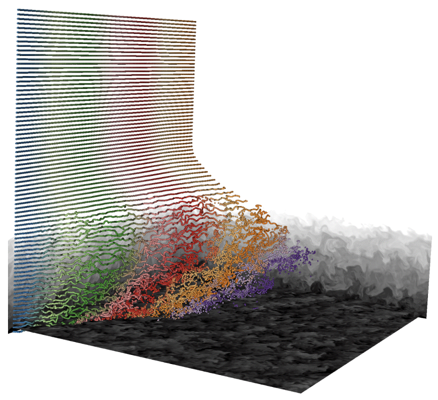


Figure 32: NekRS simulation for the atmospheric boundary layer flows demonstrating the potential temperature on the vertical planes and the streamwise velocity on the horizontal plane.

The Reynolds number is $Re = UL_b/\nu$, where $L_b = 100$ m is the thickness of the initial thermal boundary layer and ν is the molecular viscosity. Under the stated conditions, $Re \approx 50$ M, which precludes direct numerical simulation (DNS) wherein all turbulent scales are resolved.

We consider periodic boundary conditions (BCs) in the streamwise and spanwise directions. At the top boundary, ($z = 400$ m), a stress-free, rigid lid is applied for momentum, and the heat flux for the energy equation is set consistent with the 0.01 K/m temperature gradient initially prescribed in the upper region of the flow. At the bottom boundary, we use a wall model in which traction BCs for the velocity. The specified shear stress comes from Monin-Obukhov similarity theory [82]. For the energy equation, a heat flux is applied that is derived from the same theory and a specified potential temperature difference between the flow at a height, z_1, and the surface. For the traction BCs for the horizontal velocity components, we followed the approaches of [45, 63] in the context of the log-law. The normal component of the velocity as zero and the traction BCs are imposed on the tangential

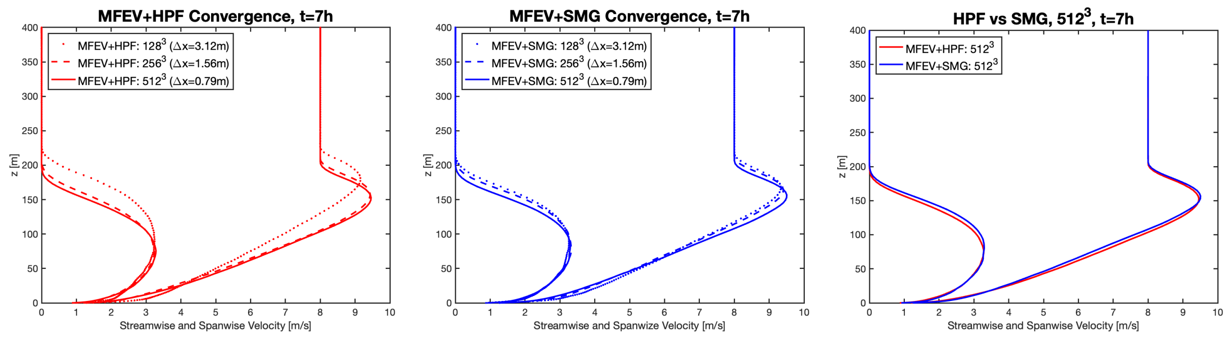


Figure 33: Horizontally averaged streamwise and spanwise velocities at $t = 7h$ with MFEV+HPF and MFEV+SMG using traction boundary conditions, demonstrating convergence at three different resolutions.

Strong-Scaling on Frontier, $n = 512^3$, $\Delta x = 0.78$ m, $\Delta t = 6.25\text{e-}2\text{s}$, $\Omega = [400m]^3$

			NekRS						AMR-Wind					
node	GCD	n/GCD	v_i	p_i	T_i	t_{step}	P_{eff}	r_t	v_i	p_i	T_i	t_{step}	P_{eff}	r_t
1	8	1.6777e+07	1	1.55	1	5.8832e-01	100	9.41	1	2	1	1.1320e+0	100	18.12
2	16	8.3886e+06	1	1.72	1	3.0980e-01	94.9	4.95	1	2	1	6.1235e-1	92.4	9.79
3	24	5.5924e+06	1	1.67	1	2.1456e-01	91.4	3.43	1	2	1	4.5652e-1	82.6	7.30
6	48	2.7962e+06	1	1.64	1	1.1294e-01	86.8	1.80	1	2	1	2.6995e-1	69.9	4.31
7	56	2.3967e+06	1	1.66	1	1.0106e-01	83.1	1.67	1	2	1	2.4527e-1	65.9	3.92
8	64	2.0971e+06	1	1.75	1	9.5191e-02	77.2	1.52	1	2	1	2.0544e-1	68.9	3.28
12	96	1.3981e+06	1	1.57	1	6.5960e-02	74.3	1.05	1	2	1	1.7106e-1	55.1	2.73
18	144	9.32067+05	1	1.67	1	5.3373e-02	61.2	0.85	1	2	1	1.4021e-1	44.8	2.24
24	192	6.9905e+05	1	1.71	1	4.6695e-02	52.4	0.74	1	2	1	1.2275e-1	38.4	1.96
48	384	3.4953e+05	1	1.65	1	3.5196e-02	34.8	0.56	-	-	-	-	-	-
96	768	1.7476e+05	1	1.64	1	3.0367e-02	20.1	0.48	1	2	1	8.2161e-2	14.3	1.31
192	1536	8.7381e+04	1	1.56	1	2.9127e-02	10.5	0.46	1	2	1	7.4918e-2	7.87	1.19

Strong-Scaling on Frontier, $n = 1024^3$, $\Delta x = 0.39$ m, $\Delta t = 3.12\text{e-}2\text{s}$, $\Omega = [400m]^3$

			NekRS					
node	GCD	n/GCD	v_i	p_i	T_i	t_{step}	P_{eff}	r_t
5	40	2.6844e+07	1	1.73	1	9.6554e-01	100	30.89
7	56	1.9174e+07	1	1.62	1	6.9376e-01	99.4	22.20
12	96	1.1185e+07	1	1.62	1	4.1021e-01	98.0	13.12
20	160	6.7109e+06	1	1.66	1	2.5999e-01	92.8	8.31
24	192	5.5924e+06	1	1.80	1	2.2913e-01	87.7	7.33
54	432	2.4855e+06	1	1.65	1	1.1270e-01	79.3	3.60
60	480	2.2370e+06	1	1.52	1	1.0345e-01	77.7	3.31
64	512	2.0972e+06	1	1.58	1	9.7153e-02	77.6	3.10
96	768	1.3981e+06	1	1.82	1	7.5584e-02	66.5	2.41
192	1536	6.9905e+05	1	1.60	1	5.1362e-02	48.9	1.64
384	3072	3.4953e+05	1	1.67	1	4.2848e-02	29.3	1.37

Table 4: NekRS GPU and AMR-Wind GPU strong-scaling on Frontier for the resolutions of 512^3 (top) and NekRS strong-scaling on Frontier using resolution of 1024^3 (bottom).

velocity based on the horizontally-averaged slip velocity that develops at the boundary. Further details regarding the wall model are provided in [78].

Newly Developed Subgrid-Scale Models and Convergence. We have explored several SGS modeling approaches for the GABLS problem including a mean-field eddy viscosity (MFEV) [112] for the anisotropic part of the stress tensor. The law of the wall is effected through the use of the MFEV concept and the approach originally used by [100] is used to convert the horizontally-averaged traction to local values based on the local slip velocity in each of the horizontal directions. The isotropic part of the dissipation is provided using either a high-pass filter (HPF) [111] or a Smagorinsky (SMG) eddy viscosity [107] applied to the

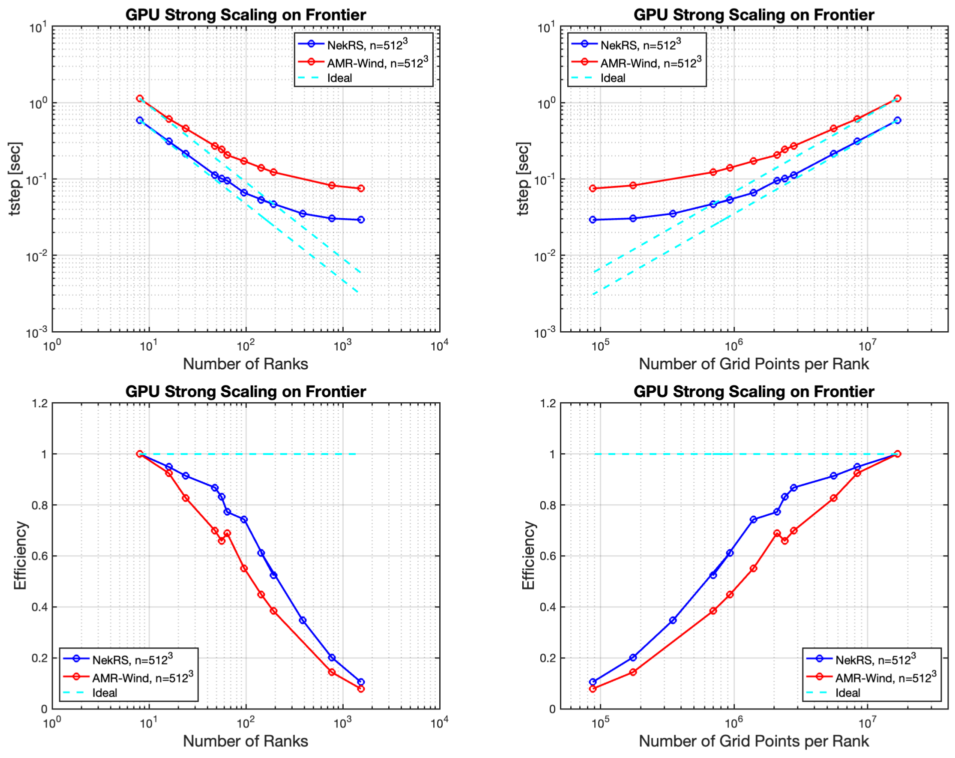


Figure 34: NekRS and AMR-Wind on Frontier, $n = 512^3$.

fluctuating strain-rate. Here, we present some of the basic components of these approaches and compare their convergence behavior. Additional thermal components required for these models are presented in [78].

For the HPF approach [111], which is not eddy-viscosity based, we have

$$\tau_{ij}^{sgs} = -2\nu_T \langle S_{ij} \rangle , \tag{6}$$

where the angle brackets $\langle\ \ \rangle$ denote averaging over the horizontal directions (at each timestep) and ν_T is an average eddy viscosity, which is expressed in terms of mean flow quantities,

$$\nu_T = (C_K L_m)^2 \sqrt{2 \langle S_{ij} \rangle \langle S_{ij} \rangle}. \tag{7}$$

Here, C_K is a constant and $C_K L_m$ is a mixing-length scale [112, 78]. In the HPF model, isotropic fluctuations are damped by addition of the $-\chi HPF(u_i)$ term to f_i, as mentioned earlier.

In our second model, we still consider the SGS dissipation effected through a non-isotropic, MFEV, but an isotropic, fluctuating part, is taken into account through a Smagorinsky (SMG) model based on the fluctuating strain rate. Our approach is extended from the SGS model of [112] based on the following expression:

$$\tau_{ij}^{sgs} = -2\gamma\nu_t S_{ij} - 2\nu_T \langle S_{ij} \rangle , \tag{8}$$

where also here the angle brackets $\langle\ \ \rangle$ denote averaging over the homogeneous directions and ν_T is an average eddy viscosity which is expressed in terms of mean flow quantities. In Eq. (8) γ is an "isotropy factor," which accounts for variability in the SGS constants due to anisotropy of the mean flow. In [112], the fluctuating eddy viscosity, ν_t, is obtained using an eddy viscosity model based on the SGS turbulent kinetic energy equation [101],

$$\nu_t = (C_s \Delta)^2 \sqrt{2 S'_{ij} S'_{ij}}, \tag{9}$$

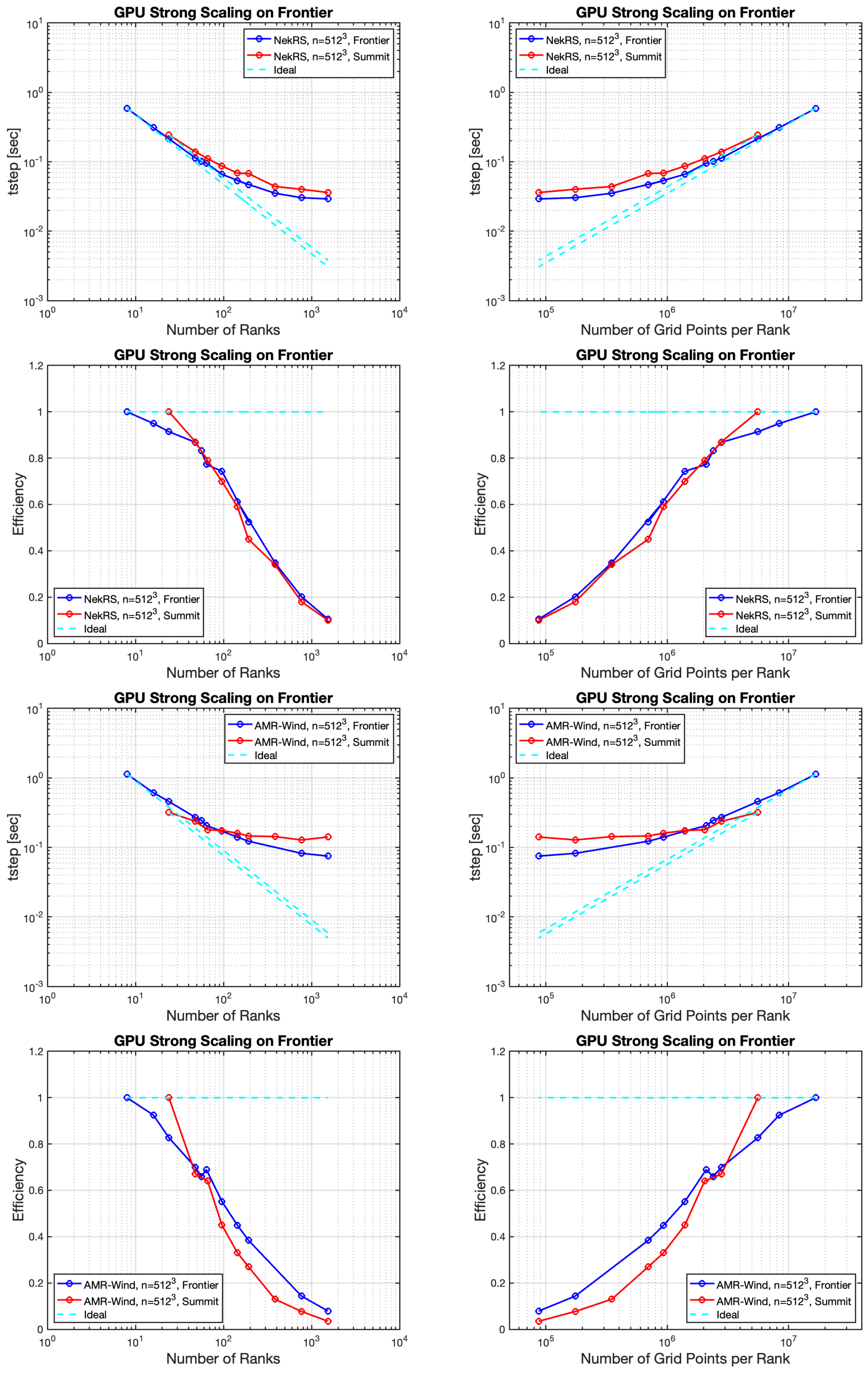


Figure 35: NekRS on Frontier vs. Summit, 512^3 (top and second rows). AMR-Wind on Frontier vs. Summit, 512^3 (third and bottom rows). The results on Summit are from our previous studies [78].

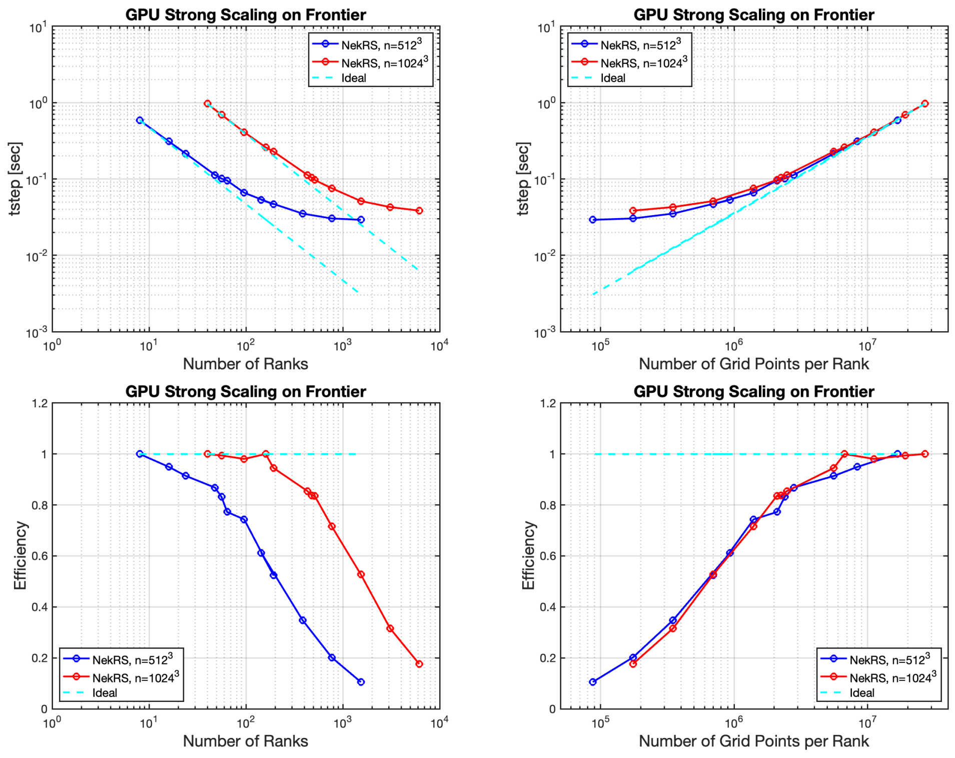


Figure 36: NekRS on Frontier, $n = 512^3$ and $n = 1024^3$.

where $S' = \sqrt{2\left\langle (S_{ij} - \langle S_{ij}\rangle)(S_{ij} - \langle S_{ij}\rangle)\right\rangle}$ and $C_s = \left(C_k\sqrt{\frac{C_k}{C_\epsilon}}\right)^{1/2}$. The isotropy factor γ is obtained by $\gamma = \frac{S'}{S' + \langle S\rangle}$ with $\langle S\rangle = \sqrt{2\langle S_{ij}\rangle\langle S_{ij}\rangle}$. (See [78, 74] for further detail.)

Figure 33 shows the horizontally averaged streamwise and spanwise velocities at $t = 7h$ at three different resolutions using $n = 128^3, 256^3, 512^3$ which correspond to the averaged grid spacing of Δx =3.12m, 1.56m, 0.79m, respectively. The top left shows relatively rapid convergence for MFEV+HPF, with a clearly converging profile at $n = 256^3$ that is very close to the one at $n = 512^3$. The results of MFEV+SMG, while not as converged at $n = 256^3$ as MFEV+HPF, shows profiles that are fairly close at all three resolutions, which implies that MFEV+SMG is perhaps more robust than MFEV+HPF. The third panel illustrates that both models converge to the same jet height, which is an important parameter for wind-farm modeling.

ExaWind Frontier Runs. CEED NekRS team and ExaWind team at NREL previously reported scaling performance of two CFD codes, NekRS and AMR-Wind, on Summit and Crusher for GABLS benchmark problem in [1,2]. In this report, we extended the studies to Frontier. We demonstrate strong scaling studies and performance analysis on Frontier, provided with comparison with previous results on Summit.

Table 4 (top) shows strong-scaling of NekRS and AMR-Wind on Frontier for the GABLS benchmark problem using the resolution of $n = 512^3$ (134M gridpoints) using 8–1536 GCDs. We used turbulent initial condition at physical time 6 hours and ran 200 timesteps, and measured t_{step} by averaging the time from 101 to 200 steps, provided with the efficiency and the ratio of simulation time to real time, r_i. NekRS shows 80% efficiency using n/P = 2M–2.3M with t_{step} = 0.09–0.1 seconds per step, whereas AMR-Wind shows 80% efficiency using n/P = 2.7M–5.5M with t_{step} = 0.26–0.45. (Here P is the MPI ranks, equivalent to the number of GCDS.) We observe that NekRS is faster than the real time with 0.05 seconds per timestep at 61% efficiency using n/P = 0.9M. These results are demonstrated in Fig. 34 using the conventional strong scaling with respect to the number of GCDs (left column) and the metric based on the number of grid points per GCD, n/P (right column).

Table 4 (bottom) shows strong-scaling of NekRS on Frontier for a larger problem size $n = 1024^3$ (1B gridpoints). We observe 80% efficiency using n/P =∼2.5M with ∼0.1 seconds per timestep.

Figure 35 shows the performance on Frontier in comparison to that on summit for NekRS (first and second rows) and AMR-Wind (third and fourth rows). The results on Summit are from our previous studies reported in [1].

Figure 36 shows NekRS performance on Frontier for two different resolutions using $n = 512^3$ and $n = 1024^3$, shown in Table 4. The right top in Fig. 36 replots the strong-scaling information with n/P as the independent variable. Here, we see a collapse of each code's (conventional) strong-scaling data into a single curve for NekRS. The efficiency plot, right bottom, clearly shows the $n/P = \sim$1.8M on Summit and 2.4M on Frontier as the 80% parallel efficiency point for NekRS.

4.5 Other ECP Applications

In addition to the main CEED ECP applications described in the previous sections, CEED researchers from the MAGMA team were also involved in helping the ExaSGD and E3SM projects in the ECP:

- The Optimizing Stochastic Grid Dynamics at Exascale (ExaSGD) application is focused on reliable and efficient planning of the power grid, optimize the grid's response to many potential disruption events under different weather scenarios. In all simulations, ExaSGD employed the LDLt solvers from CEED's MAGMA library. Notably, MAGMA stands out as the sole stable numerical linear algebra solution for the ExaSGD KPP2 computations.

- The Energy Exascale Earth System Model (E3SM) application is an ongoing, state-of-the-science Earth system modeling, simulation, and prediction project that optimizes the use of DOE laboratory resources to meet the science needs of the nation and the mission needs of DOE. The E3SM land model, spearheaded by Dali Wang and Peter Schwartz from ORNL, has been adapted for NVIDIA and AMD GPUs. The initial Fortran code utilized LAPACK's dgbsv routine to address a band diagonal matrix challenge (a compact matrix of size 7x15 with 5 bands in the compressed band format) for each land grid cell. In its initial GPU adaptation, OpenACC was employed to offload this LAPACK function, allowing for the parallel solving of 12,000 band diagonal matrices (encompassing a total of 6,000 land grid cells, with two matrices each) on a singular GPU. However, this approach yields less than optimal performance due to data transfers between the CPU and GPU. To address this, CEED's MAGMA team developed and released in MAGMA a batched band solution, mirroring LAPACK's dgbsv routine, which runs entirely on the GPU. This batched computation is highly optimized for GPUs and, furthermore, eliminates the need for GPU-CPU data transfers [3].

5. EXTERNAL APPLICATIONS

In addition to select ECP applications the key CEED software products of MFEM, NEK, libCEED, MAGMA, libP, OCCA, PUMI and Omega_h are being used by a growing user community external to the ECP. This includes DOE and non DOE users including universities, research organizations and industry.

The subsections that follow provide a small sample of applications using or being built on CEED products.

5.1 Physics Equation Translator for MFEM (Petra-M)

- Application Name: Physics Equation Translator for MFEM (Petra-M)
- Application Owner: Syuníchi Shiraiwa, Princeton Plasma Physics Laboratory, shiraiwaprinceton.edu
- Application URL: https://github.com/piScope/PetraM_Base
- Sponsor(s): Department of Energy through the offices of Fusion Energy Systems (FES) and Advanced Scientific Computing Research
- CEED Products Used: MFEM, PUMI, Omega_h
- Brief Application Description: An electro-magnetic (EM) simulation tool for modeling RF wave propagation

Petra-M (Physics Equation Translator for MFEM) is an integrated finite-element-method (FEM) open source multi-physics platform [102, 104]. Petra-M was originally developed to provide a graphical user interface to configure FEM RF simulations using MFEM.

The Petra-M framework continues to be expanded towards a full featured self-contained general purpose FEM analysis platform, which allows for (i) creation of arbitrary geometry, (ii) mesh generation, (iii) efficient assembly of finite element linear systems for a variety of physics partial differential equations (PDEs), (iv) matrix solution with direct or iterative solvers, and (v) results visualization. Petra-M combines the variety of open source libraries, a large fraction of which are being developed by Advance Scientific Computing Research (ASCR) including Parallel Unstructured Mesh infrastructure (PUMI) (https://scorec.rpi.edu/pumi/index.php) for mesh adaptation, Hypre (Scalable Linear Solvers and Multigrid (http://llnl.gov/casc/hypre)), MUMPS (A Parallel Sparse Direct Solver (http://mumps.enseeiht.fr/)), and STRUctured Matrix PACKage (Strumpack) (https://portal.nersc.gov/project/sparse/strumpack/) for the linear solver and MFEM for the FEM linear system assembly. Furthermore, the combination OpenCASCADE (https://www.opencascade.com/) and GMSH (http://gmsh.info) or modules of the Simmetrix Simulation Modeling Suite are used for geometry and mesh generation. Finally, the user can set up a simulation starting from the geometry generation to the visualization of the numerical solutions via a GUI user interface [103], which is an open-source python-based data analysis environment.

PUMI based mesh adaptation techniques that have been integrated into MFEM [106] and are being applied to large scale simulations of RF antenna in tokamak fusion systems. Figure shows the electric field solved for on a adaptively refined mesh of just under one million elements Figure 37.

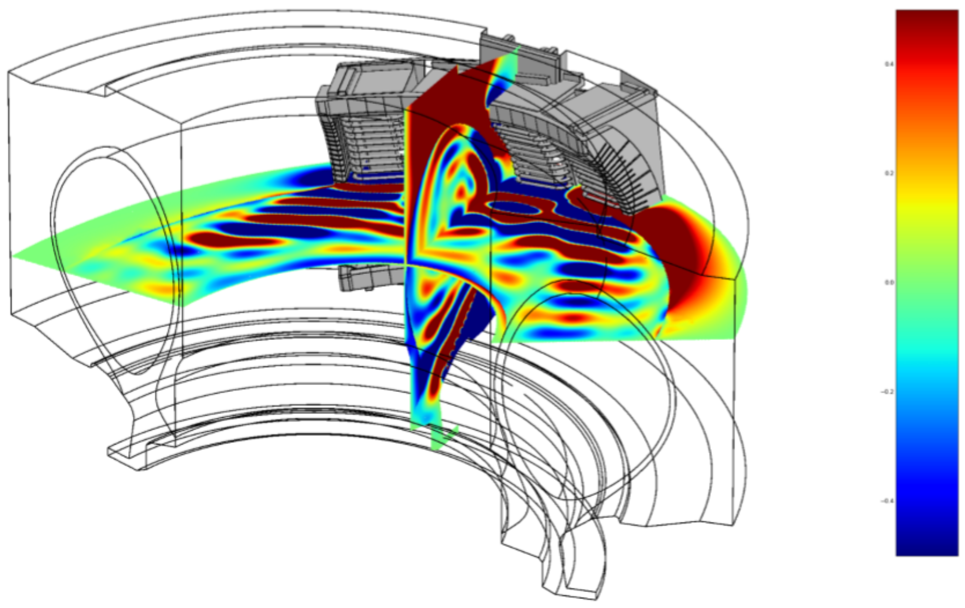


Figure 37: Electric field solved for on an adapted mesh of 4th order elements.

5.2 Multi-physics Integrated Simulations and Optimization (MISO)

- Application Name: Multi-physics Integrated Simulations and Optimization (MISO)
- Application Owner: Jason Hicken, Rensselaer Polytechnic Institute, hickej2@rpi.edu
- Application URL: https://github.com/OptimalDesignLab/MISO
- Sponsor: NASA
- CEED Products Used: MFEM

- Brief Application Description: MISO is a library for partial-differential-equation-based simulation and optimization of aerospace systems. MISO uses MFEM to simulate multi-physics problems — such as thermal-electromagnetic motor analyses — and couples these simulations with NASA's OpenMDAO framework to enable multidisciplinary design optimization.

5.3 CEED-PHASTA

- Application Name: CEED-PHASTA
- Application Owner: Jed Brown, Jed.Brown@Colorado.EDU and Kenneth E. Jansen, Kenneth.Jansen@colorado.edu both at University of Colorado Boulder
- Application URL: https://libceed.org/en/latest/examples/fluids/
- Sponsor(s): DOE, NSF, NASA
- CEED Products Used: libCEED, PETSc, MAGMA, PUMI
- Brief Application Description: Unstructured grid, compressible Navier-Stokes solver employing (SUPG/VMS) stabilized finite elements and implicit time integration. While the same discretization approach has been used successfully on CPUs under the PHASTA open source project, this application makes use of CEED backends that are not only still efficient on CPUs but are very efficient and scalable to GPUs.

5.4 River Dynamical core for E3SM (RDycore)

- Application Name: RDycore: River Dynamical core for E3SM
- Application Owner: Jed Brown Jed.Brown@Colorado.EDU and Gautam Bisht <gautam.bisht@pnnl.gov>
- Application URL: https://github.com/RDycore/RDycore/
- Sponsor(s): DOE
- CEED Products Used: libCEED, PETSc
- Brief Application Description: Unstructured grid finite volume solver for shallow water dynamics in compound flooding (pluvial, fluvial, and storm surge) with support for anisotropic meshes and implicit/explicit integration. RDycore is being developed as a dynamical core for E3SM to be used in century-scale climate models and for high-resolution regional modeling. Efficiency and scalability of the CPU and GPU modes have been demonstated on Perlmutter and Crusher.

5.5 Framework for Antarctic System Science in E3SM (FAnSSIE)

- Application Name: FAnSSIE: Framework for Antarctic System Science in E3SM
- Application Owner: Matthew Hoffman, Los Alamos National Laboratory, mhoffman@lanl.gov and Mauro Perego, Sandia National Laboratories, mperego@sandia.gov
- Application URL: https://fanssie.github.io/
- Sponsor(s): SCIDAC BER, EESM, ASCR
- CEED Products Used: Omega_h
- Brief Application Description: FAnSSIE creates an Antarctic system science capability for DOE's E3SM climate model by adding key missing processes for the ice sheet, ocean, and snowpack, and the coupling between them. GPU accelerated unstructured mesh adaptation and mesh motion provided by Omega_h is key to tracking evolving land ice sheet boundaries and to capture physical fields of interest.

5.6 PArallel LArge-scale Computational Electromagnetics (Palace)

- Application Name: PArallel LArge-scale Computational Electromagnetics (Palace)
- Application Owner: Sebastian Grimberg, AWS Center for Quantum Computing, Grimberg, sjgamazon.com
- Application URL: https://github.com/awslabs/palace
- Sponsor(s): Amazon Web Services (AWS)
- CEED Products Used: MFEM, libCEED
- Brief Application Description: Cloud-based electromagnetics simulation tool for quantum computing hardware

Palace is a parallel finite element code for full-wave electromagnetics simulations developed at the Amazon Web Services (AWS) Center for Quantum Computing to perform large-scale 3D simulations for the design of quantum computing hardware. It supports a wide range of simulation types: eigenmode analysis, driven simulations in the frequency and time domains, and electrostatic and magnetostatic simulations for lumped parameter extraction. As an open-source project, it is also fully extensible by developers looking to add new features for problems of industrial relevance. An example of a Palace-based simulation result can be seen in Figure 38.

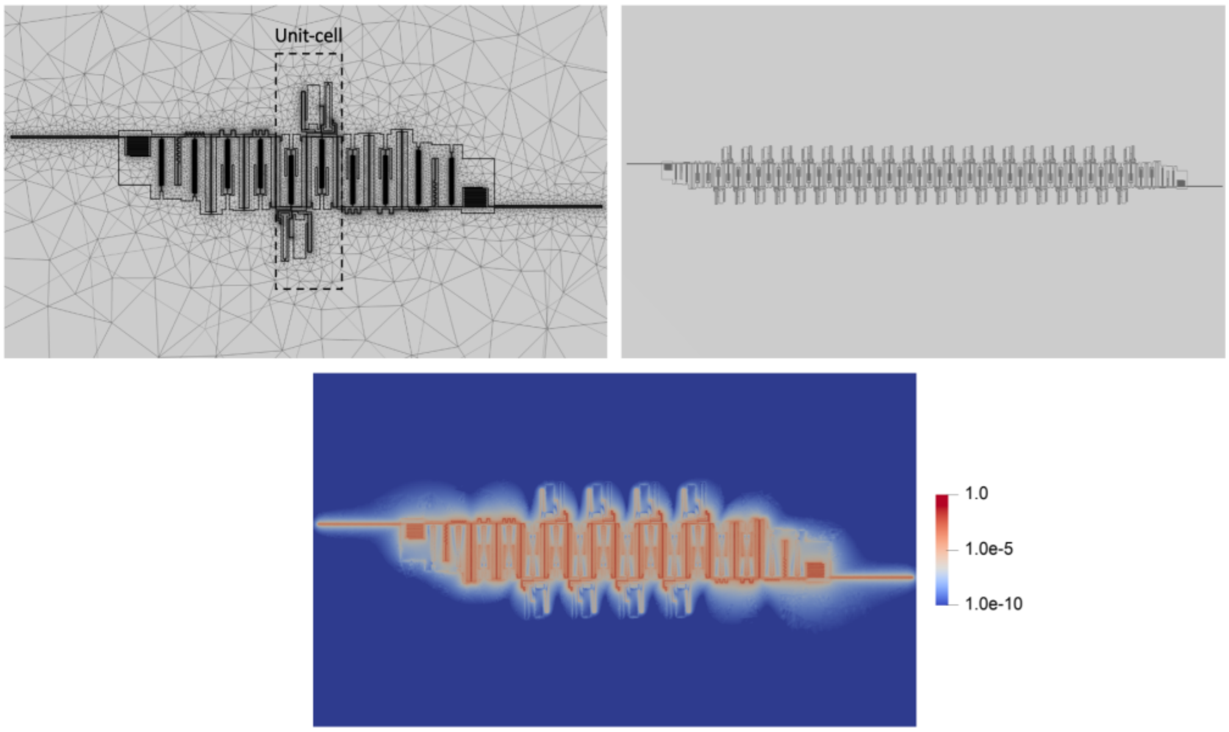


Figure 38: 1, 4, and 21 unit-cell repetition models of a metamaterial waveguide simulation, with engineered tapers at both ends. The 4 unit-cell repetition (bottom) visualizes the electric field energy density, scaled by the maximum over the entire computational domain, from the solution computed by Palace at 6 GHz.

Palace is built on top of CEED's MFEM finite element discretization library, which provides parallel mesh data structures supporting adaptive mesh refinement, high-performance arbitrary high-order finite element spaces, scalable linear solvers, and more. In recent work, MFEM's programming abstractions for single-source support of CPU and GPU device execution and integrated memory management have enabled the porting of

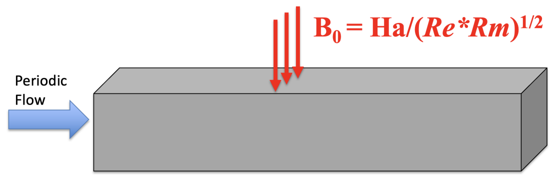


Figure 39: Configuration for laminar Hartmann flow in square duct.

Palace to GPU systems. Palace also utilizes libCEED to take advantage of the partial assembly framework for efficient storage and application of high-order finite element operators, again on modern hardware including various GPU backends. libCEED's optimized CPU and GPU backends for non-tensor-product elements enable Palace's efficient matrix-free p-multigrid-based linear solvers for simulation models based on simplex and mixed meshes.

5.7 Incompressible Magnetohydrodynamics (NekRS)

- Application Name: MHD
- Application Owner: Misun Min, Argonne National Laboratory, mmin@mcs.anl.gov
- Sponsor(s): CEED
- CEED Products Used: NekRS
- Brief Application Description: Turbulent MHD simulations at Exascale

We have developed an incompressible MHD solver into NekRS solving the following equations:

$$\frac{\partial \mathbf{u}}{\partial t} - \frac{1}{Re}\nabla^2\mathbf{u} + \nabla p = \mathbf{B}\cdot\nabla\mathbf{B} - \mathbf{u}\cdot\nabla\mathbf{u}, \tag{10}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{11}$$

$$\frac{\partial \mathbf{B}}{\partial t} - \frac{\tilde{\sigma}}{Rm}\nabla^2\mathbf{B} + \nabla q = \mathbf{B}\cdot\nabla\mathbf{u} - \mathbf{u}\cdot\nabla\mathbf{B}, \tag{12}$$

$$\nabla\cdot\mathbf{B} = 0, \tag{13}$$

where $\mathbf{u}$ is the fluid velocity, $\mathbf{B}$ is the magnetic field, p is the fluid pressure, and q is the magnetic pressure. The governing equations (10)–(13) involve solving two linear Stokes subproblems per timestep with coupling by advection-like terms on the right hand sides, which means we can reuse most of the features (fast kernels, scalable communication, etc) existing in Nek5000/RS solvers. Here additional complication is that the domain for $\mathbf{B}$ is larger than the domain for $\mathbf{u}$ so that the boundary conditions for $\mathbf{B}$ can be imposed farther away. While NekRS is prepared for conjugate heat transfer problem with solid and fluid domains (the solid domain is typically larger than the fluid domain), it was not tested for MHD.

Here our NS/MHD splitting scheme involves three steps: (i) explicit: tentative velocity from BDFk/EXTk applied to nonlinear terms, (ii) implicit pressure Poisson solve + projection onto divergence-free space, (iii) implicit viscous update (which retains $\nabla\cdot\mathbf{u}$ to order of accuracy). For the explicit update, only, we can simplify the problem by switching to Elsasser variables defined by

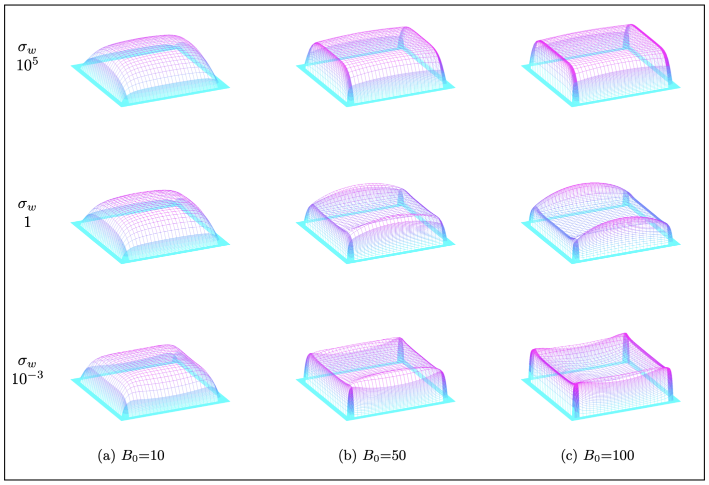
$$z^+ = \mathbf{u} + \mathbf{B}, \quad z^- = \mathbf{u} - \mathbf{B}. \tag{14}$$

In this case, the nonlinear terms in the explicit updates simplify to:

$$RHS^+ = -z^-\cdot\nabla z^+, \quad RHS^- = -z^+\cdot\nabla z^-, \tag{15}$$

which means we have just two convection terms to evaluate instead of four.

We consider testing problems for an impulsively started flow in a square duct at low Reynolds number, with applied magnetic field B_0, as illustrated in Fig. 39 so that flows go only one direction. Figure 40



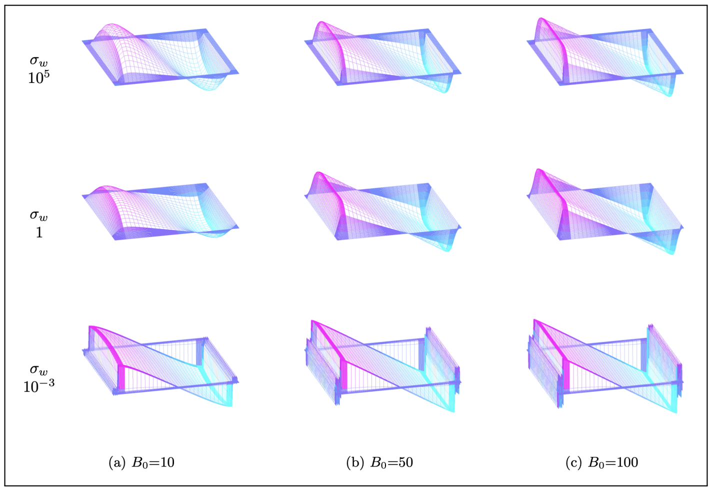


Figure 40: Hartmann model problem: (Top box) velocity distribution and (Bottom box) magnetic field distribution with applied magnetic field strength of $B_0 = 10$, 50 and 100. Here $\tilde{\sigma} = 1$ on the fluid domain Ω_F and $\tilde{\sigma} = \sigma_w$ is the relative magnetic diffusivity on the side wall $(\Omega_S - \Omega_F)$ for the solid domain Ω_S.

demonstrates the velocity and magnetic field profiles for Hartmann problem. Our Hartmann flows were started by impulsively applying a mean pressure gradient at $t = 0$. As a result, the solution exhibits a ringing response, where the flow oscillates in the axial direction as it decays to the steady-state solution as shown in Fig. 41. We can estimate this response by looking at the most slowly-decaying eigenmode of the MHD system, which involves sines and cosines in the x-y directions. The detailed studies can be found in [48].

5.8 Particle Tracking Simulations (NekRS)

- Application Name: Particle Tracking
- Application Owner: Misun Min, Argonne National Laboratory, mmin@mcs.anl.gov
- Sponsor(s): CEED
- CEED Products Used: Nek5000
- Brief Application Description: Efficient particle tracking simulations for dense particle systems.

We consider large-scale dense particle systems from PPICLF (parallel particle-in-cell library in Fortran) developed by S. Balachandar and D. Zwick and explore numerical algorithms that can simulate accurately

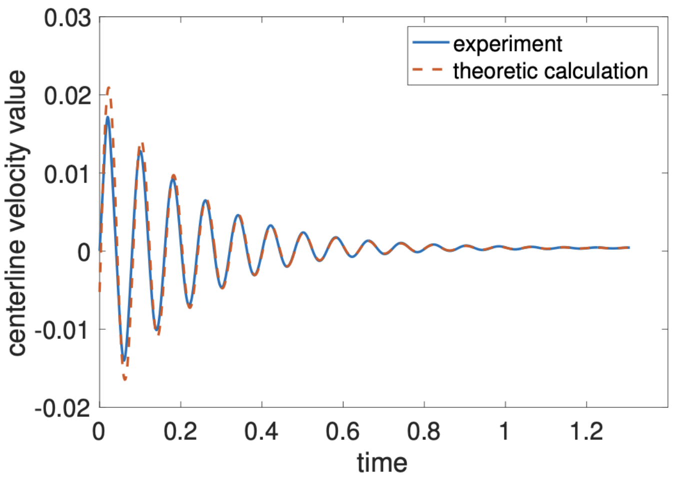


Figure 41: Hartmann model problem: Time history of center-point velocity for $Ha = 50$, $Re = 1$, and $Rm = 1$.

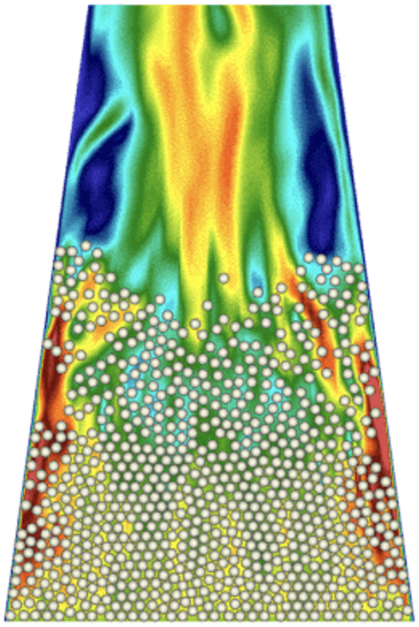


Figure 42: Nek5000 four-way coupling particle tracking simulations.

and efficiently. In particular, we consider uniform distribution of dense particles in various geometries and extend semi-implicit timestepping algorithms from Nek5000 into PPICLF for one-, two- and four-way coupling representing fluid-particle, particle-fluid, particle-particle interaction [87]. We provide convergence studies and demonstrate simulation capabilities using various examples. Figure 42 demonstrates four-way coupling simulation in a slanted box shape with dense particles.

5.9 Treble Technologies Acoustics Solver for Architectural Design

- Application Name: Acoustics Solver for Architectural Design
- Application Owner: Treble Technologies
- Application URL: https://www.treble.tech

- Sponsor(s):
- CEED Products Used: libParanumal and OCCA CEED
- Brief Application Description: Treble Technologies develops sound simulation technology and processes.

Treble Technologies develops state-of-the art software for acoustic simulations. Such simulations have a wide variety of use cases ranging from room acoustics [95] and audio hardware design [114] to oil exploration [83] and whale tracking [51]. At Treble, the emphasis is on the room acoustics and audio applications where high fidelity simulations are of great value.

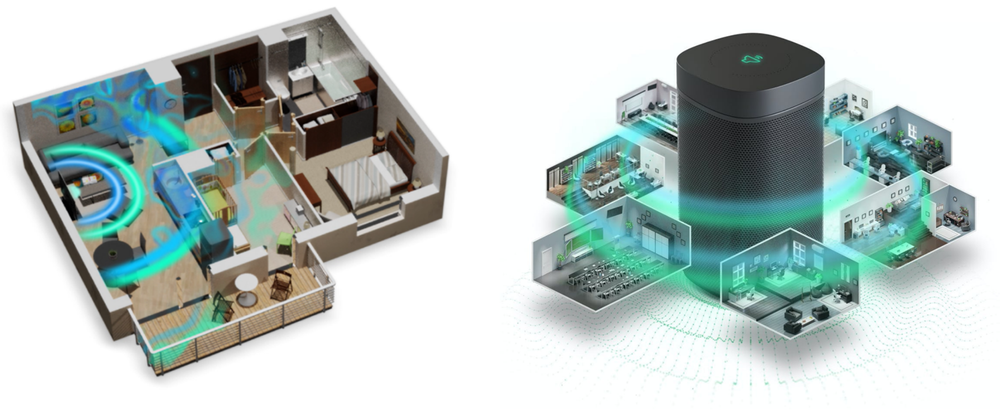


Figure 43: Acoustic waves attached to two of the main focuses of Treble.

Simulation algorithms: Treble's simulation algorithms can be split into two categories. A geometrical one (ray-based) and a numerical wave-propagation one (wave-based). The latter one is the core of Treble's simulation technology, but it is computationally heavy and requires sophisticated highly-parallel algorithms to even be usable for realistic use cases.

The underlying simulation technology of the wave-based solver is called the nodal discontinuous-Galerkin method (DGM) [53]. The DGM splits its computations into multiple local elements with element boundary terms communicating fluxes between elements. Decomposing the wave problem into a collection of local elements yields favorable conditions for GPU-acceleration.

At Treble, we use these methods to solve the acoustic wave-equation which can be represented by a coupled system of equations

$$\frac{\partial \mathbf{v}}{\partial t} = -\frac{1}{\rho}\nabla p, \tag{16}$$

$$\frac{\partial p}{\partial t} = -\rho c^2 \nabla \cdot \mathbf{v}, \tag{17}$$

where $\mathbf{v}$ represents particle velocity, t time, p pressure, and ρ and c represent the density and acoustic velocity in air, respectively. Solving such a problem grows in compute requirements with the resolved frequency of the waves to the power of four meaning that resolving high-frequency wave-propagation is computationally demanding. The audible spectrum ranges from 20 Hz to 20.000 Hz so the simulations need to cover a wide frequency spectrum, meaning that at the high end the simulations become computationally demanding requiring high-performance computing.

The libParanumal and OCCA CEED libraries are used in our DGM solver to facilitate discretization and GPU-acceleration. It is common that simulations require large mesh files which need to be distributed between multiple GPUs along with the creation of ghost layers for information exchange between GPUs, all of this is handled efficiently by libParanumal. Thanks to these two libraries, we only need to think about the mathematics and physics, while the parallelization and GPU implementation is abstracted away.

5.10 Shell Performance Portable Seismic Inversion

- Application Name: (p)SWELL
- Application Owner: Shell Global
- Sponsor(s):
- CEED Products Used: OCCA
- Brief Application Description: Shell Global performance portability platform for seismic imaging.

Overview. Adapting high-performance software to various architecture while ensuring performance is a challenging endeavor more so in the energy industry where HPC is at the inception of explorations and production activities. Seismic exploration is not feasible without seismic imaging (Reverse Time Migration: RTM) and velocity model building (Full Waveform Inversion: FWI) which both entirely rely on supercomputers. With the recent advent of large offset and low frequency seismic data, the data acquired in surveys has become ever richer and more voluminous. At the same time, a push for more detailed solutions requires the inclusion of higher frequencies from the data. Moreover, to support extracting accurate and realistic geophysical models of the subsurface, velocity model building such as done in FWI frequently requires inclusion of anisotropic parameters, elastic, and viscous information. In this work, we briefly describe how we ported our existing proprietary seismic libraries to GPUs and how this effort will work out for many other architectures.

HPC software portability. Adapting or, even better, rapidly adapting to emerging compute architectures is non-trivial. Various constraints, not only technical, for porting codes to different architectures must be considered:

- Size of existing body of code.
- Desired performance.
- Ease of software migration.
- Developer skills and cost.
- Risk/reward.

The topic of fully re-writing codes can be perilous in any context, and, most often the preferred route is to undertake a carefully planned incremental refactor of existing libraries and algorithms. A common activity nowadays consists in porting code from a latency-based architecture (CPUs) to a throughput based one (GPUs). Many vendors now propose viable GPU or off-load compute solutions: NVIDIA, AMD, NEC, Xilinx and Intel all have their own approach. Hence, the ability to port once and use on many architectures would be ideal. Achieving this seamlessly directly through standard languages and compilers is still not possible at the time of writing. To aid this process, there's a plethora of libraries and compiler directive based approaches available to help migrate code to different architectures see (non-exhaustive list) [85], [30], [118], [39], [84], [46], [121], [16]. For the CPU to GPU migration, two main solutions exists, both compiler directive based, the well-established OpenMP [118] and a more recent approach called OpenACC [30]. Nevertheless, it is still very difficult to unleash full performance compared with native vendor APIs. In the case of OpenACC. For instance, in an early in-house experiment, only 60% of CUDA performance was achieved using the PGI compiler suite with special pragmas enabling shared memory usage. Similar observations were made in [34] and ideas from [96] could not completely palliate to the performance losses. Another weakness point of OpenACC is a lack of support across vendors. Intel does not support OpenACC while AMD and NVIDIA both support it. Libraries such as Kokkos [39] or SYCL [46] require very good skill level in C++, something not available in all groups. A lesser known open source alternative is the Open Concurrent Computing Abstraction or OCCA [71, 108, 109]. It offers most of the options of the mainstream languages, but it also allows the use of native vendor APIs. Again, in early in-house experiments with OCCA, performance matching one of the CUDA kernels was achieved. OCCA uses an abstraction for the programming of devices, a C extension called OKL (OCCA Kernel Language) which is designed to be vendor agnostic. OKL gets translated in the native vendor APIs at runtime. The main vendor API supported are CUDA (NVIDIA), HIP

(AMD) and DPC++ (Intel). Those can 1 2 3 be accessed either directly or through the OKL layer. OCCA is mainly a Just-In-Time (JIT) approach where compute kernels are compiled when and where they are used. JIT compilation of many kernels can be problematic. To remedy this, the packaging of often used kernels in libraries is feasible (the runtime will look in the file for precompiled kernels) or simply pre-populating the OCCA kernel cache with often used kernels is also a possibility. It supports lambda expression, can be embedded in Fortran, C and C++ codes. More importantly, even with this ability to achieve excellent performance, it is simple to use and understand.

Approach. Using OCCA we have refactored the libraries at the heart of the wave equation modeling in our seismic codes (e.g. FWI, RTM, LS-RTM) [120], [69]. Most seismic imaging or velocity building codes can be summarized as follows:

- A forward time-loop, with injection of source data (forcing term) propagated, using a form of the wave equation, where snapshots for the entire simulation domain are saved (usually in compressed format [ZFP:19]) to disk or parts of the memory hierarchy.
- A backward time-loop doing almost the same as the forward loop but where the data saved to the disk in the previous step is either correlated or used in a gradient calculation either in-line (during back-propagation) or offline.

The project had tight deadlines and the time given to launch the new code in production on a GPU cluster was 12 months. Starting from zero lines of GPU code and none of the middle-ware / infrastructure adapted to handle GPU workloads, it took about one year to have the FWI code able to run on GPUs in a production setting. The crux of the FWI code underwent almost no changes with the refactors occurring almost exclusively in the libraries. To be fair, the code had been correctly designed to adapt to various plug-in libraries recently. Moving to GPUs vindicated the quality of the approach. Our approach in porting the libraries required modifications of three main aspects:

- Finite-difference kernels (FD) (propagators),
- Traces / injection / extraction (forcing terms),
- IO, compression, snapshotting (snapshots).

Overall, for CPU kernels, the OpenMP mode was chosen and, for GPU kernels, CUDA was elected: since the first target platform was NVIDIA hardware. A subset of CPU and GPU kernels were later translated to OKL. Which means, for CPU like architectures, one OKL kernel is compiled in OpenMP mode while, for GPU like architectures, another OKL kernel (written differently) is employed. Hence, to cover most of the landscape of available CPU and GPUs, only two sets of kernels need to be maintained for parts of the code in need of acceleration. However, each kernel needs to be tuned (e.g. blocking, thread block sizes) for every CPU and GPU families. FD kernels solving the wave equation are known and built at start-up of the simulation and used for many time-steps. For FD, the classical approach of [17] [18] was followed. Other kernels are done in-place, which means, when first encountered, a compilation and run occurs while any later time the kernel is ran from cache. How trace data is injected and extracted from the smaller GPU memory is performed with two kernels an “inject” and a "record" one. The JIT enables to know the parameters for the reduction at compile time improving performance and simplifying invocation [19]. Moreover, recent OBN surveys use many source points at large offsets yielding sometimes more than a million of long (in-time) traces (after use of the so called reciprocity theorem) that require more memory than is available on a GPU. A double buffering approach for transferring traces to the CPU removes such limitations. Since the GPU FD kernels are much faster than the CPU ones, the snapshots required during the gradient calculation needed to be revisited. The IO can now overlap with computation for many GPU and CPU FWI setups. This is performed using a thread pool on the CPU and using streams running parallel to the compute stream for device to host transfers. Compression, necessary to manage the data volumes, either on the CPU or GPU, is also integrated. Two libraries are used for compression ZFP [19] and Bitcomp from NVIDIA. Even though many opportunities for acceleration of this GPU capable FWI still exists, the current approach enabled to accelerate the workflows many times.

Conclusions and future work OCCA enabled Shell to extend hardware support of its low-level wave propagation and IO libraries. Those libraries are used in the preferred algorithms in production seismic processing. The libraries support all know forms of the first-order form of the wave equation (acoustic, elastic) with all possible anisotropies up to 21 coefficients in the elasticity tensor and has support for isotropic viscosity. The overall effort future proofs our architecture agnostic libraries and therefore our algorithms for years to come.

6. CEED-6.0 SOFTWARE 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. The distribution is tested on several platforms including Linux (RHEL and Ubuntu), Mac OSX, and the representative heterogeneous HPC machines OLCF Crusher/Frontier, LLNL Tioga and Lassen, with some packages tested on ALCF Sunspot. See `https://ceed.exascaleproject.org/ceed-6.0/` for details about native installation using Spack and containerized use via Docker, Singularity, and Shifter.

Version CEED-6.0 released on September 30, 2023 contains 16 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-5.0, some of which were reported in [60]; the list below reflects only the latest releases since CEED-5.0.

FMS-0.2 (no change)

GSLIB-1.0.6 (no change)

Laghos-3.1 (no change)

libCEED-0.12 (planned) New features in this release include:

- SYCL backend for Intel GPUs
- Support for OpenMP parallelism
- Improved `memcheck` debugging features
- Support for basis evaluation at particles, for use with material-point methods.
- Support for non-tensor $H(\text{div})$ and $H(\text{curl})$ spaces.
- New subgrid stress model for the fluids example.

MAGMA-2.7.2 (August 25, 2023) New additions in this release include expert interfaces for LU, QR, and Cholesky factorizations; introduced tuning specifications for LU, QR, and Cholesky factorizations; enhanced tuning for Ampere and later GPUs; integrated a fused LU panel for AMD GPUs; addressed bug issues related to batched LU on singular matrices. See the MAGMA 2.7.2 release notes for further details. Additionally, in our continuous effort to port and optimize MAGMA for Intel GPUs, the MAGMA team has been actively updating a public MAGMA branch with SYCL support, gearing up for an upcoming release.

MFEM-4.6 (September 27, 2023) New additions in this release include:

- NURBS meshing and discretization improvements.
- SubMesh support for H(curl) and H(div) transfers.
- TMOP enhancements and mesh optimization miniapps.
- New H(div) matrix-free saddle-point solver.
- Ultraweak DPG formulation for acoustics and Maxwell.
- Stochastic PDE method for Gaussian random fields.

- Obstacle problem and topology optimization examples.
- k-d tree for parallel grid function repartitioning.
- HIP support in the PETSc and SUNDIALS interfaces.
- Improved GPU code debugging.
- and much more!

For more details, see the interactive documentation at https://mfem.org and the full CHANGELOG.

Nek5000-19.0 (no change)

NekRS-23.0 (May 30, 2023) New additions in this release include:

- Lagrangian phase model (one-way coupling)
- Overset grids (neknek)
- Particle tracking
- Single source udf+oudf
- Device support BoomerAMG
- Improved runtime statistics
- 4th-kind Chebyshev smoothers
- Configureable time averaging
- Extrapolation initialGuess method
- Scaleable JIT compilation
- Real gas support for lowMach
- More examples
- Various bug fixes

See the NekRS-23.0 release notes for further details.

Nekbone-17.0 (no change)

NekCEM-c8db04b (no change)

OCCA-1.6.0 (August 22, 2023) New additions in this release include:

- Enhanced multithreading support on the HOST side.
- Quality of life improvements for developers
- Support for typedef enums and enums in OKL kernels.

See the changelog for this release.

Omega_h-scorec-v10.7.0 (September 23, 2023) Adds support for a Kokkos backend with the only use of vendor APIs (i.e., AMD HIP, NVIDIA CUDA, Intel SYCL/DPL) for stable sorting on GPUs. An implementation of a Kokkos-based memory pool is also included. See Section 2.3 for details.

PETSc-3.20 (September 28, 2023) Changes relevant to CEED include:

- `PetscDeviceContextGetStreamHandle` allows sharing stream/queues between PETSc (including Kokkos) and third-party libraries (such as libCEED) without exposing stream/queue types in the public interface.
- Revamped profiling is more flexible and enables both traditional logging and tracing for flamegraphs.
- Variable-size point-block Jacobi `vpbjacobi` preconditioner setup now runs on GPUs.
- Refreshed STRUMPACK support.

- Expanded support for libCEED via DMPlex.
- 4th kind Chebyshev smoothers.

PUMI-2.2.8 (September 21, 2023) Adds support for Gmsh v4 files, a parallel CGNS reader and writer, and support for extruded meshes in the Simmetrix conversion utility, and improvements for the PHASTA CFD pre processor. See the release notes for details.

Ratel-0.3 (planned) New features in this release include:

- Update strain energy function to the convex form for both Neo-Hookean and Mooney-Rivlin models.
- Add mixed linear elasticity and Neo-Hookean hyperelastic models for incompressible materials.
- Add linear plasticity with linear hardening model for small strains.
- Add pressure boundary loading which is caused by liquids or gases on the surface of the solid structure.
- Add flexible clamp, slip, traction, and platen boundary conditions with more complex time variance during a quasistatic or dynamic simulation.
- Add robust surface force and surface centroid monitoring options.
- Add command-line validation options for maximum displacement and per-face centroids and surface forces.
- Add isochoric Ogden hyperelastic model in initial configuration.
- Add mixed Ogden hyperelastic model in initial configuration.
- Add isochoric Neo-Hookean model in initial configuration.
- Add Mooney-Rivlin hyperelastic model in current configuration.
- Add Coulomb friction capabilities to platen contact boundary conditions.
- Add isochoric Mooney-Rivlin model in initial configuration.
- Add CEED benchmark problems 1, 2, 3, and 4 for convergence testing and benchmarking.
- Setup dynamic solver for mixed linear elastic and hyperelastic models.
- Setup performance and memory footprint optimizations.

Remhos-1.0 (no change)

7. ADDITIONAL TOPICS

This section covers additional research performed by the CEED team on topics such as mesh optimization, solvers, implicit fluid simulations, performance tuning on NVIDIA GPUs and exascale simulation workflow.

7.1 CEED's Educational Impact

The CEED project has had a significant educational impact both through its research and software products (some of which are used to teach classes at universities, e.g., MFEM in RPI, libCEED at CU Boulder) as well as through the numerous summer interns, graduate students and post-docs that participated in its activities. Brief bios of these researchers are provided below.

- Dr. Anthony Austin joined the CEED team as a postdoc at Virginia Tech in 2018. He contributed to the linear solvers and PDE solver in the libParanumal software package. He co-authored a SISC article on initial guesses for linear solvers using stable sparse and least squares extrapolation methods [12]. After his postdoc Dr. Austin joined the Naval Postgraduate School in Monterey as an Assistant Professor of Mathematics in 2019.

- Adeleke Bankole was supported through the CEED project as a postdoc at CU Boulder, where he worked on discretizations and local Fourier analysis for high-order methods applied to wave propagation and fluids.
- Valeria Barra was supported through the CEED Project as a postdoc at CU Boulder, where she made numerous contributions to libCEED and developed new mini-apps for PDEs on the sphere and for compressible viscous flow. She is now a Research Software Engineer at Caltech, working on the CliMA project.
- Dr. Pedro Bello-Maldonado, currently a staff research scientist at IBM Research, was supported by CEED during his Ph.D. studies at UIUC, where he developed scalable preconditioners for Poisson problems, including a novel approach to FEM-based preconditioners for the spectral element method [19], which forms the core of the SEMFEM preconditioner in Nek5000/RS.
- Dr. Noel Chalmers joined the CEED team as a postdoc at Virginia Tech in 2017. During his time on the CEED project he assumed a leadership role on the libParanumal software project making considerable contributions at every level. Dr. Chalmers has appeared as a co-author on numerous papers resulting from his CEED related algorithmic contributions [29, 113, 57, 12, 58] and software contributions [43, 25, 28]. After his postdoc Dr. Chalmers joined AMD Research in Austin, Texas in 2018. At AMD Research he has added a HIP backend to the OCCA portability library, enabling all code that are built using OCCA to use AMD GPUs. He also developed a new AMD internal benchmark code (hipBone) based on CEED BP5 and libParanumal [27]. Finally, he has also played a significant role in the first HPL calculations to exceed an exaflop on the Frontier system at ALCF [26, 67].
- Professor Som Dutta was supported by CEED as post-doctoral researcher in the CS department at UIUC working on scalable multiphase flow simulations. He is currently an assistant professor in the mechanical engineering department at Utah State. His group makes extensive use of Nek5000/RS for particle tracking applications, including aerosol tracking.
- Leila Ghaffari was partially supported through the CEED project as a PhD student at CU Boulder. Her work included verification and validation for compressible viscous flow and hyperelasticity, derivation of new material models with Enzyme's algorithmic differentiation, and algebraic solvers for singular nonlinear mechanics problems.
- Yichen Guo was partially funded through the CEED project as a graduate student at Virginia Tech. She continues to develop new solvers based on libParanumal and she recently submitted a paper to SISC on stopping criteria that better balance numerical discretization error and linear solver error when using iterative methods to solve CEED bake-off problems [47]. She was also funded through the CEED project with an appointment as 2023 Summer Intern at Argonne National Laboratory, focusing on developing an incompressible MHD solver into NekRS [48].
- Aditya Joshi was partly supported through the CEED Project as a Ph.D. student focused on implementing GPU based curved element, conforming mesh adaptation in MFEM.
- Dr. Ali Karakus joined the CEED team as a postdoc at Virginia Tech in 2016. During his time on the CEED project he developed an incompressible Navier-Stokes solver as part of the libParanumal project. His CEED related research has resulted in numerous co-author credits including [113, 57, 58, 43]. After his postdoc at Virginia Tech Dr. Karakus took a postdoc in MCS at Argonne National Lab where he continued to develop the incompressible flow solver. Subsequently he joined the Middle Eastern Technical University (METU) in Turkey as an Assistant Professor of Mechanical Engineering. Dr. Karakus has mentored numerous graduate students at METU and many have used libParanumal as a starting point for their research.
- Yu-Hsiang Lan was funded through the CEED project as an M.S. student in CS at UIUC and as a Visiting Student and Summer Intern at Argonne National Laboratory. His work focused on scalable steady state solvers, extreme-scale test and development on advanced GPU architectures for Nek5000/RS [77, 76, 72], and development of all-hex meshing tools for full-core simulations of pebble-bed reactors

[65]. Yu-Hsiang has been responsible for pushing Nek5000/RS to the limit through extensive full-scale tests on DOE's leadership computers, Summit and Frontier.

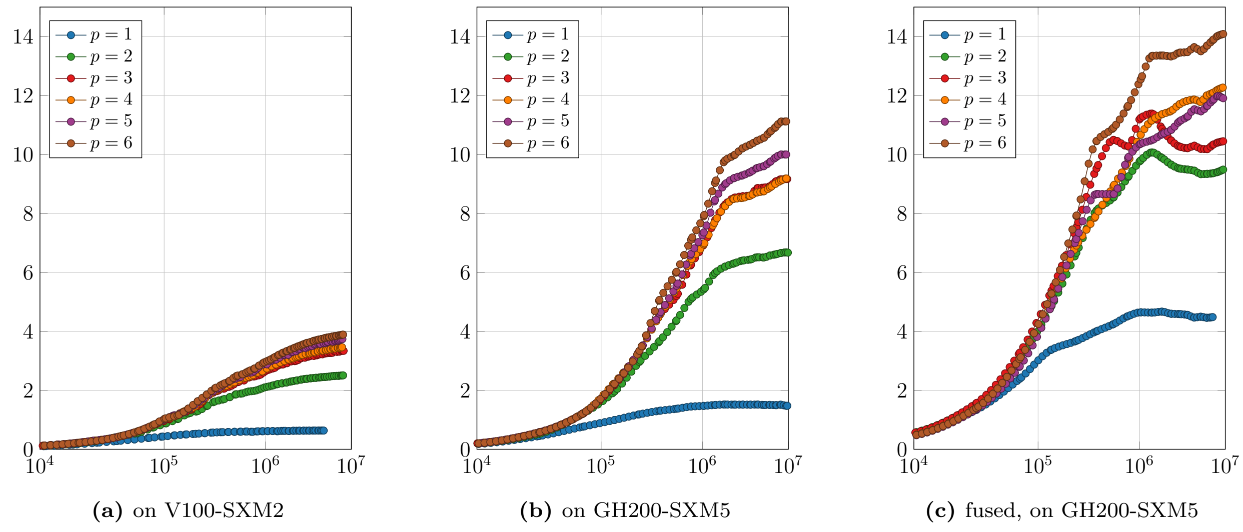
- Yimin Lin was supported by the CEED project as a 2020 Summer Intern at Argonne National Laboratory during his Ph.D at Rice University, focusing on developing a molten-salt-reactor simulations based on the coupled Navier-Stokes and Poisson-Nernst-Planck (PNP) equations.
- Dr. Neil Lindquist was supported by the CEED project as 2020 Summer Intern at Argonne National Laboratory during his Ph.D at University of Tennessee Knoxville. He developed fast scalable interpolation on GPUs [66] (a GPU-based version of *findpts*) to support particle tracking and overset grids for various applications including thermal fluids, bubble dynamics, and aerosol transport.
- Matthew McCall was supported through the CEED Project as an undergraduate student in computer science and designed and implemented the Kokkos based memory pool for Omega_h.
- Aditya Parik is fully funded through the CEED project with an appointment as 2023 Summer Intern at Argonne National Laboratory during his Ph.D. at Utah State, focusing on particle tracking simulations for dense particle systems.
- Dr. Malachi Phillips was fully funded under CEED as a Ph.D. student in the CS department at UIUC. In addition to being a lead developer of NekRS, Malachi made several novel developments in high-order preconditioners, including tuned Chebyshev-accelerated Schwarz smoothers for p-multigrid [91], 4th-kind Chebyshev smoothing with optimal V-cycles [93], and SEMFEM for NekRS. Malachi has been a frequent intern at Sandia National Laboratories, where he now is a post-doctoral researcher.
- Thilina Rathnayake is fully funded through the CEED project as a Ph.D. student at UIUC. He worked extensively on NekRS development and performance analysis, focusing on exascale workflows, scalable partitioners, programming models, and highly-scalable coarse-grid solvers. Thilina has also been a summer intern at Argonne, LLNL, and Intel.
- Dr. Morteza Siboni was partly supported through the CEED Project as a RPI postdoc and then Research Scientist. He implemented the PUMI based mesh adaptation procedures in MFEM and applied them to RF problems [105].
- Dr. Kasia Swirydowicz joined the CEED team as a postdoc at Virginia Tech in 2017. During her postdoc she developed novel algorithms for the CEED bake-off kernels [113]. After her postdoc she initially took a position at NREL on the ExaWind project and is currently a Computational Scientist at Pacific Northwest National Laboratory.
- Jeremy L. Thompson was supported through the CEED Project as an Applied Math PhD student at CU Boulder. Jeremy took a leading role in the development of libCEED and his thesis work also included local Fourier analysis of high-order operators. He is now a Research Software Engineer for CU Boulder's PSAAP center, developing material-point methods in Ratel with libCEED and PETSc.

7.2 MFEM Hopper GH200 Benchmark Results

The benchmarks were carried out on one Grace-Hopper GH200-SXM5 superchip. The total amount of global memory is 92 GBytes. There are 132 multiprocessors each with 128 CUDA cores. The CUDA driver 12.3 and runtime 12.2 were used to gather the results presented in figure 44.

Figure 44 presents the results of the CEED BP1 (mass) in terms of throughput (GDOF/s), depending on the number of degrees of freedom (DOF):

- Figure 44a is on one Volta V100 on Lassen,
- Figure 44b is on one Grace-Hopper GH200-SXM5, it is the first time that this benchmark exceeds 10 GDOF/s on a single-chip.
- Figure 44c is on one Grace-Hopper GH200-SXM5, with the benchmark being entirely fused in one kernel. Besides the improved $N_{0.8}$ already presented in the CEED report MS37, this Hopper chip also exhibits a peak rate of work per unit resource ($r_{\max}$) better than the non-fused standard kernel.



(a) on V100-SXM2 **(b)** on GH200-SXM5 **(c)** fused, on GH200-SXM5

Figure 44: Performance of the CEED BP1 benchmark on one Grace-Hopper GH200-SXM5, in throughput (GDOF/s), depending on the number of degrees of freedom (DOF)

7.3 Multidomain Support for Overset Grids in NekRS

We have recently extended NekRS to support multiple overset grids, which significantly enhances geometric flexibility when simulating flow in complex domains and or past multicomponent structures with moving parts. Rotating machinery, for example, cannot be treated with ALE (arbitrary Lagrangian Eulerian) formulations because of mesh tearing. By contrast, such configurations are easily treated with overset grids in which one subdomain moves with respect to another.

Overset grids are effectively an extension of Schwarz overlapping methods to the full Navier-Stokes equations. Rather than iterating between domains on individual substeps of the time advancement, we have each domain make a full Navier-Stokes step, using time-extrapolated data on portions of the boundary that intersect with another domain. (Iteration is used on the full *linear* Stokes substep, when needed for stability.) A key requirement for an efficient implementation is to have a fast and scalable general-purpose interpolation utility that can provide function values at arbitrary points in the computational domain. In this way, once can extract velocity/pressure values wherever a given domain boundary lies within the overlap region of a neighboring subdomain. In [66] we reported on fast GPU-based extensions of *findpts()*, which is a highly-scalable general purpose interpolation routine tailored to spectral elements.

This recent effort builds upon earlier CPU-based work of Ketan Mittal [80], in which each subdomain is assigned to a separate MPI rank, thereby ensuring a private memory space for each domain. The new implementation extends the developments of Lindquist [66] to include several features that are necessary when supporting more than two subdomains. As noted by Mittal [79], these include resolution of interpolation "ownership" when multiple domains overlap, mass conservation across subdomain interfaces, and partition-of-unity weight functions for computing integrals over the full domain. An important benefit of applying Schwarz to the full Navier-Stokes equations is that one can readily use different timestep sizes in different domains, as described in [81]. Implementation of this feature in NekRS will be supported in a future release.

7.4 Stopping Criteria

We consider the Poisson problem

$$-\nabla \cdot (\kappa(x)\nabla u(x)) = f(x) \tag{18}$$

on a bounded domain $\Omega \subset \mathbb{R}^d$. We impose homogeneous Dirichlet boundary conditions $u = 0$ on $\partial\Omega$. Here $f \in L^2(\Omega)$. Further, we assume there exists a constant α such that $0 < \alpha \le \kappa(x) \in L^2(\Omega)$.

Let V_h represent the finite element space of piecewise polynomials of degree N on the triangulation $\mathcal{T}$. We define s as the dimension of V_h and denote by ϕ_n basis functions of V_h. Additionally, we refer to $\mathcal{E}$ as the set of all $(d-1)$-dimensional element edges (faces in $\mathbb{R}^3$) of $\mathcal{T}$. The finite element approximation to (18) is: find $u_h \in V_h$ such that

$$\int_\Omega \kappa(x)\nabla u_h \cdot \nabla \phi_n \, dx = \int_\Omega f\phi_n \, dx, \quad \forall \, n = 1, \cdots, s. \tag{19}$$

The approximation problem 19 is equivalent to the linear system:

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{20}$$

Solving the linear system by an iterative method, we obtain $\mathbf{x}_k \in \mathbb{R}^s$ as the approximate solution to 20 at the k-th iteration which in turn provides an approximate finite element solution $u_h^k = \sum_{n=1}^s x_n^k \phi_n$.

As a result, the total error, $\|u - u_h^k\|$, arises from two main sources: discretization error $\|u - u_h\|$ and algebraic error $\|u_h - u_h^k\|$. As the iteration proceeds, the algebraic error gradually decreases to zero, causing the total error to converge to the discretization error. Ideally, the iteration should be terminated once the discretization error is dominant in the total error. However, the stopping criterion based on relative residual norm, $\|\mathbf{r}_k\| \leq \tau\|\mathbf{r}_0\|$, where $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$, lacks information about discretization, which might induce numerous unnecessary iterations. As an illustrative example discussed in [47, Section 4.2], 45 shows that after the stagnation of the total error, the criterion requires nearly eighty additional iterations, which contributes negligibly to improving the accuracy of finite element solution. For the purpose of both terminating the iterative solver as early as possible and maintaining the accuracy of the finite element solution, the design of stopping criteria in finite element frameworks has been explored in numerous papers [8, 11, 94, 56, 10, 40, 9, 86].

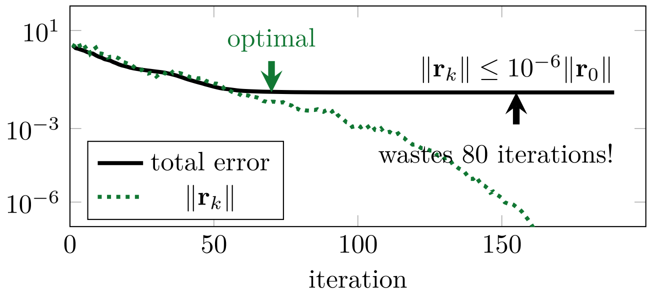


Figure 45: Stopping criteria based on relative residual norm might waste numerous iterations.

In this work [47], we introduce a new stopping criterion. The criteria, balancing algebraic and discretization errors, can be derived directly from the linear residual. The n-th component of the linear residual $\mathbf{r}_k$ is

$$\begin{aligned}(\mathbf{r}_k)_n &= \mathbf{b}_n - (\mathbf{A}\mathbf{x}_k)_n \\ &= (\phi_n, f) - \sum_{K\in\mathcal{T}_h} \left(\kappa(x)\nabla\phi_n, \nabla u_h^k\right)_K.\end{aligned}$$

Integrating the last term by parts, we obtain

$$(\mathbf{r}_k)_n = \sum_{K\in\mathcal{T}_h} \left(\phi_n, f + \nabla\cdot\left(\kappa(x)\nabla u_h^k\right)\right)_K - \sum_{\ell\in\mathcal{E}} \left(\phi_n, \left[\left(\kappa(x)\nabla u_h^k\right)\cdot \mathbf{n}_\ell\right]\right)_\ell,$$

where $[\mathbf{u}\cdot\mathbf{n}_\ell]$ is the jump of the normal component of $\mathbf{u}$ across the edge ℓ and $\mathbf{n}_\ell$ is the outward normal vector. We introduce vectors $\mathbf{R}_k, \mathbf{F}_k \in \mathbb{R}^s$ with the components given by

$$(\mathbf{R}_k)_n = \sum_{K\in\mathcal{T}_h} \left(\phi_n, f + \nabla\cdot\left(\kappa(x)\nabla u_h^k\right)\right)_K \text{ and } (\mathbf{F}_k)_n = \sum_{\ell\in\mathcal{E}} \left(\phi_n, -\left[\left(\kappa(x)\nabla u_h^k\right)\cdot\mathbf{n}_\ell\right]\right),$$

respectively. Additionally, similar to the $\kappa(x)$ scaling in [21], we define the weighted l^2-norm of $\mathbf{x}$ as $\|\mathbf{x}\|_\mathbf{w} = \left(\mathbf{x}^T\mathbf{W}\mathbf{x}\right)^{1/2}$. Here, $\mathbf{W}$ is a diagonal matrix with entries given by

$$\mathbf{W}_{n,n} = \min_{x\in\omega_n} \kappa(x)^{-1},$$

where $\omega_n = \operatorname{supp}(\phi_n)$.

With the above considerations, we propose an indicator η_{RF},

$$\eta_{\mathrm{RF}} := \|\mathbf{R}_k\|_{\mathbf{w}} + \|\mathbf{F}_k\|_{\mathbf{w}}, \tag{21}$$

with the associated stopping criterion:

$$\|\mathbf{r}_k\|_{\mathbf{w}} \le \tau\, \eta_{\mathrm{RF}}.$$

The indicator η_{RF} provides an upper bound for the **w**-norm of the residual without any unknown constants involved. Ideally, η_{RF} closely tracks $\|\mathbf{r}_k\|_{\mathbf{w}}$ until the total error converges, and the separation between $\|\mathbf{r}_k\|_{\mathbf{w}}$ and η_{RF} should indicate the deviation of the total error from the algebraic error. Furthermore, it is only necessary to compute $\mathbf{R}_k$; $\mathbf{F}_k = \mathbf{r}_k - \mathbf{R}_k$ can be directly calculated once $\mathbf{R}_k$ has been determined.

To demonstrate the effectiveness of the stopping criterion,we consider the Poisson equation on a L-shaped domain with highly variable piecewise constant coefficients. As shown in 46, the domain Ω is partitioned into four subdomains and $\kappa(x)$ is constant on each subdomain. We solve the linear system by the conjugate gradient algorithm and set $\tau = 1/20$. In 47, the total error becomes stagnant around 140th iterations. During the initial iterations, the indicator η_{RF} follows $\|\mathbf{r}_k\|_{\mathbf{w}}$. The separation of η_{RF} and $\|\mathbf{r}_k\|_{\mathbf{w}}$ coincides with the stagnation of the total error, halting the iteration at a reasonable point, as highlighted by the arrow.

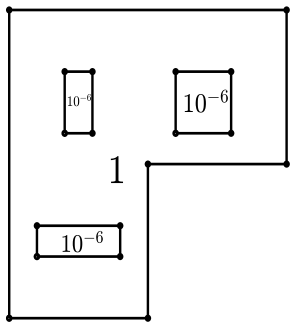


Figure 46: Domain Ω and coefficient $\kappa(x)$.

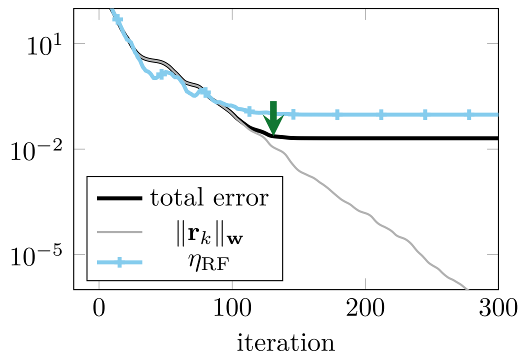


Figure 47: Convergence history with $N = 6$.

To measure the reliability of a stopping criterion, we define the quality ratio of a criterion as

$$\text{quality ratio} := \frac{\|u - u_h^{k^*}\|}{\|u - u_h\|}, \tag{22}$$

where $u_h^{k^*}$ is the first solution that satisfies the stopping condition during the iterative process. 5 presents numbers of iterations and quality ratios resulting from applying stopping criteria to the solution with $N = 4, 6, 8$. In contrast to the criterion based on the relative residual norm, the proposed criterion terminates iterations much earlier. Meantime, the small quality ratios indicate the reliable accuracy of the finite element solution.

Table 5: Numbers of iterations (iter) and quality ratios (qual. (22)).

Criterion	$N=4$		$N=6$		$N=8$	
	iter	qual.	iter	qual.	iter	qual.
$\|\mathbf{r}_k\|_{\mathbf{w}} \le \tau\eta_{\mathrm{RF}}^{\mathbf{w}}$	70	1.13	131	1.14	201	1.26
$\|\mathbf{r}_k\| \le 10^{-6}\|\mathbf{r}_0\|$	149	1.00	256	1.00	390	1.00

7.5 Fast Coarse Grid Solvers

We consider development of coarse grid solvers tailored to exascale architectures for the pMG preconditioner [68, 6, 54, 92] used in the solution of the pressure Poisson problem that arises when simulating unsteady incompressible flow with spatial discretization based on the Spectral Element Method (SEM) [33, 89]. Pressure Poisson equation is usually solved in a form similar to the following at each timestep of an incompressible flow simulation:

$$-\nabla^2 \tilde{u} = f \ \text{ in } \Omega \tag{23}$$

with boundary conditions $\tilde{u} = 0$ on $\partial\Omega_D$ and $\nabla\tilde{u} \cdot \hat{n} = 0$ on $\partial\Omega_N$, where $\partial\Omega_D$ and $\partial\Omega_N$ are respective Dirichlet and Neumann boundaries. For SEM or finite element methods (FEM), the discretization is based on the variational formulation: *Find* $u \in X_0^N \subset \mathcal{H}_0^1$ *such that*

$$a(v,u) = \int_\Omega vf \, dV, \qquad \forall v \in X_0^N, \tag{24}$$

where the inner product $a(v,u)$ and function space $\mathcal{H}_0^1$ are defined as:

$$\mathcal{H}_0^1 = \left\{ v \,\middle|\, \int_\Omega \nabla v \cdot \nabla v \ < \infty, \ \int_\Omega v^2 \ < \infty, \ v = 0 \text{ on } \partial\Omega_D \right\} \tag{25}$$

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dV \tag{26}$$

We also define a-norm associated with the $\mathcal{H}_0^1$ as:

$$\|u\|_a = \sqrt{a(u,u)} = \left[\int_\Omega \nabla u \cdot \nabla u \, dV \right]^{\frac{1}{2}}. \tag{27}$$

The discretization of (24) starts with the choice of the finite-dimensional approximation space, $X_0^N \subset \mathcal{H}_0^1$, and a corresponding basis. For the SEM, functions in X_0^N are expressed as the sum of Lagrange interpolating functions $\phi_i(\mathbf{x})$ satisfying $\phi_i(\mathbf{x}_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta function and the $\mathbf{x}_j$'s are the function nodal points,

$$u(\mathbf{x}) = \sum_{j=1}^{n} u_j \phi_j(\mathbf{x}). \tag{28}$$

Out of all functions in X_0^N, u minimizes the error $\tilde{u} - u$ in $a-$norm defined in (27). Approximating u and v by expansion of the basis functions ϕ_i as in (28) and variational form (24) using a suitable quadrature rule, we get the linear system $A\underline{u} = \underline{b}$ where:

$$A_{ij} = a(\phi_i, \phi_j), \ \ b_i \ = \ (\phi_i, f). \tag{29}$$

Iterative methods like GMRES [99] and conjugate gradients are used to solve $A\underline{u} = \underline{b}$ with multilevel preconditioners to accelerate the solution process. For the SEM, pMG is widely used as a preconditioner for these iterative methods. pMG consists of series of L nested spectral element spaces with $l = 1$ the coarsest mesh and $l = L$ the finest mesh. The discretization approach across the levels is essentially unchanged, save for the order of the basis functions, which usually vary as $N_{l+1} = 2N_l$ with $N_1 = 1$ and $N_L = N$. If $l > 1$, a pMG sweep usually involves smoothing followed by restriction to the next coarser level when going "down" the V-cycle, or smoothing followed by prolongation (interpolation) to the next finer level when going up. With the tensor-product basis of the SEM, all operators can be applied element wise with fast tensor-contractions expressed as matrix-matrix products [33]. The method is efficient since tensor-product operations require only $O(N_l^3)$ memory references and $O(N_l^4)$ work per element for meshes with $l > 1$. These structured (tensor-product) work on the local meshes accounts for the majority of the floating-point operations (flops) and solving (or relaxing on) the communication-intensive unstructured element-vertex mesh (i.e., $l = 1$ or $N_1 = 1$), referred to as the *coarse grid*, accounts for the majority of the communication since the coarse grid system, denoted by $A_c\underline{u}_c = \underline{b}_c$ requires all-to-all communications as the inverse A_c^{-1} is completely full.

It is well known that the *coarse grid solve* is challenging in a parallel setting when the number of processes, P, is large since the coarse grid solve is the only operation in the solution of elliptic partial differential equations (PDEs) whose cost *increases* with P [41]. As P increases, the coarse grid solve invariably sets the limits on strong-scaling, which directly impacts time-to-solution in many applications. For pMG on tensor-product element meshes with E elements, the coarse grid size is $n_c \approx E$. This has become a major bottleneck since current exa- and pre-exascale platforms routinely enable simulations with $E = 10^8$–10^9 resulting in very large coarse grid systems. The importance of the coarse grid solve for exascale applications is illustrated in [73], where it accounts for 45% of the flow simulation time when running on $P = 27\,648$ Nvidia V100 GPUs on the Summit supercomputer at Oak Ridge National Laboratory. In this case, the Navier-Stokes problem has $E = 98$ million elements of order $N = 8$ leading to $n = 51$ billion degrees of freedom (or DOFs) and the coarse solve in the pMG-preconditioned pressure Poisson problem uses a single V-cycle of BoomerAMG [52].

We present an approach to reduce communication for the $N = 1$ coarse solve $A_c\underline{u}_c = \underline{b}_c$ on P processors by introducing a low-communication two-level overlapping Schwarz smoother with a novel non-nested coarse space as an alternative to AMG as a coarse grid solver in pMG preconditioners. The key ingredients of the proposed method are:

- Overlapping or non-overlapping domain decomposition using a parallel graph partitioner like parRSB [88].
- Local additive Schwarz (AS) to smooth the Level 1 error on each processor.
- A non-nested coarse grid space for the Level 1 problem, comprising $\approx P$ DOFs.

We start by partitioning the original coarse gird Ω into P non-overlapping sub-domains Ω_p $(p = 1, \dots, P)$ using parRSB [88]. Even though the sub-domains are non-overlapping, some DOFs on the sub-domain boundaries are shared between different sub-domains and has to be handled accordingly during the sub-domain solve to ensure the continuity of the solution. Let $\overline{\Omega}_p$, $1 \le p \le P$ be the extended domain that include the original elements in Ω_p and the set of elements that share at least one DOF with the element boundary $\partial\Omega_p$. For the local subdomain problem, Dirichlet boundary conditions are applied on $\partial\overline{\Omega}_p$.

The main idea of Schwarz methods is to solve a local problem in each domain and then combine the solutions to generate the approximate solution. The preconditioner is realized by defining a restriction operator R_p from functions in $X^1(\Omega)$ to $X^1(\Omega_p)$ or, in other words, from vectors in $\mathbb{R}^n$ to vectors in $\mathbb{R}^{n_p}$, where $n = \dim(X^1(\Omega))$ and $n_p = \dim(X^1(\Omega_p))$. We use order 1 finite element basis functions for the discretization of $A_c\underline{u}_c = \underline{b}_c$ on subdomains, which is represented by superscript 1 in function space X^1. The local AS preconditioner with only the local subdomain solves can then be written as:

$$M_{AS,1}^{-1} = \sum_{p=1}^{P} R_p^T A_p^{-1} R_p, \text{ where } A_p = R_p A R_p^T. \tag{30}$$

The Level 1 overlapping systems A_p, $1 \le p \le P$ in Eq (30), (size $= O(E/P) \approx 8000$ based on the strong scaling limit of the GPU based systems like Frontier and Summit) are highly localized and well suited to parallel computation yielding full P-fold parallelism. In our implementation, we use CHOLMOD [31], a fast sparse Cholesky solver to solve the local systems A_p locally on each processor.

Bounded convergence of the local AS preconditioner is achieved by adding a coarse grid correction. Assume we have a low-dimensional coarse space Y^1 =span $\{\Phi_1,\ \Phi_2,\ \dots, \Phi_{n_r}\}$ associated with a set of basis functions, $\Phi_j(\mathbf{x})$ that can represent "smooth" (i.e., low wave number) functions on Ω (we will show how we construct this space soon). Let J be an interpolation (prolongation) operator that maps coarse-space coefficients u_j^r, $j = 1, \dots, n_r$ to their fine-scale (Level 1) counterparts, u_i^c (Eq (28) with $N = 1$), $i = 1, \dots, n_c$. Because the unknowns associated with A are grid-point values (the basis for X^1 is a set of Lagrangian interpolants on nodal points $\mathbf{x}_i$), we have

$$(J)_{ij} = \Phi_j(\mathbf{x}_i). \tag{31}$$

Following the standard Galerkin formulation, which ensures that our coarse approximation in Y^1 is the best fit in a-norm, the coarse-grid solution is:

$$\underline{u} = J A_r^{-1} J^T \underline{b}, \ \ A_r := J^T A J. \tag{32}$$

With this global coarse correction the two-level Schwarz method becomes

$$M_{AS,2}^{-1} = M_{AS,1}^{-1} + J A_r^{-1} J^T. \tag{33}$$

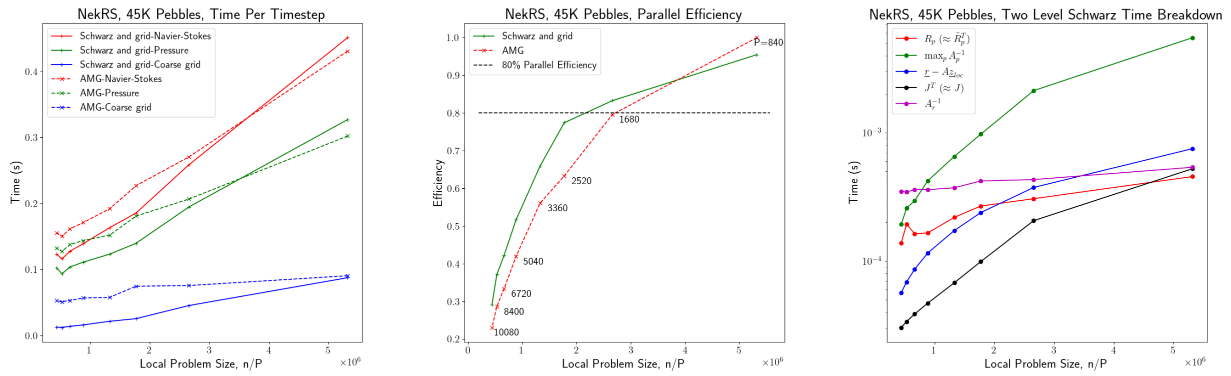


Figure 48: Strong scaling study of 45000 pebbles mesh with NekRS. Left to right: Navier-Stokes, Pressure and Coarse grid solve time per timestep for AMG and two level Schwarz method; Parallel efficiency for AMG and two level Schwarz method; Breakdown of the coarse grid solve time for the two level Schwarz method.

Identifying a coarse grid is a significant challenge for multilevel methods on unstructured grids. Common approaches like aggregation, smoothed-aggregation, and fine-coarse (FC) splittings only use the algebraic information of the system. In the PDE context, one also has important geometric information associated with the system; namely, that the basis coefficients u_i represent the unknown solution at particular points in $\mathbb{R}^d$ where d is the space dimension. With this information, one can construct spaces of low-energy functions based on simple structured grids. Let $\Omega_r \supset \Omega$ be a box-shaped coarse domain having $E_r = E_x \times E_y \times E_z$ elements in 3D or $E_r = E_x \times E_y$ elements in 2D. We define tri (or bi-) linear interpolants on these box meshes. The nodal values at the vertices of the boxes are the new coarse-space unknowns. With J the coarse-to-fine interpolation matrix defined earlier, the governing coarse-space equation is derived using the standard Galerkin approach as $A_r = J^T AJ$. This non-nested (reduced) coarse space requires only a few DOFs per process (denoted by γ) and the corresponding reduced system $A_r \underline{u}_r = \underline{b}_r$ (size $n_r = \gamma P = O(P) \ll E$, $\gamma = 1$–10), is solved using the communication-minimal XX^T algorithm developed in [44, 117]. Thus, this new method reduces the coarse grid system size from $O(E)$ to $O(P)$.

Figure 48 shows timing data for a strong scaling study of the new two level Schwarz preconditioner and AMG using 45000 pebbles mesh with NekRS [42]. For the AMG, we used BoomerAMG [52] which is the standard coarse solver in NekRS. The plots were generated by collecting timing data by increasing the number of processes, P, from 840 to 10080 (or decreasing local problem size). We can see that Navier-Stokes solve time, pressure solve time and coarse grid solve time per timestep goes down faster with the two level Schwarz solver compared to AMG as the number of processes increase in the leftmost plot in Figure 48. Crossover point where the two level Schwarz solver becomes faster than the default AMG approach is $\approx 3.5 \times 10^6$ DOFs. Middle plot of Figure 48 shows the parallel efficiency for the two solvers as local problem size decreases. Reference point for the parallel efficiency is the NekRS run with BoomerAMG solver with $P = 840$ which is assumed to have 100% parallel efficiency.

Right most plot in Figure 48 shows the breakdown of the coarse grid solve time for the two level Schwarz solver. Cost of the operations which don't involve communication goes down noticeably as local problem size decreases. These include the local Schwarz solve (A_p^{-1}), right hand side update ($\underline{r} - A\underline{z}_{loc}$, $\underline{z}_{loc} = M_{AS,1}^{-1}\underline{r}$) required for doing reduced solve in a multiplicative manner and interpolation (J) from the original coarse space A_c to the reduced structured space A_r. Note that in the case of J and J^T, it is the structured nature of A_r which allows us to compute the interpolation and prolongation in parallel with no communication. The fact that the local Schwarz solve is the most expensive operation in the two level Schwarz for most of the range presents an opportunity for further optimizations. Cost of solving the reduced system A_r^{-1} stays more or less the same as local problem size decreases. This is not surprising since the size of this system is fixed for the entire study.

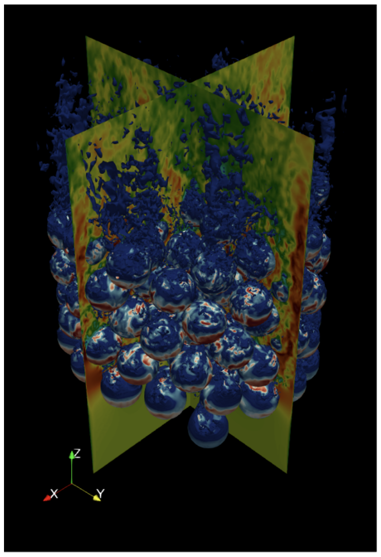


Figure 49: NekRS In situ visualization using Ascent for simulating 146 pebbles.

7.6 NekRS In-Situ Visualization

In the realm of Computational Fluid Dynamics (CFD), the demand for memory and computation resources is extreme, necessitating the use of leadership-scale computing platforms for practical domain sizes. This intensive requirement renders traditional checkpointing methods ineffective due to the significant slowdown in simulations while saving state data to disk. As we progress towards exascale and GPU-driven High-Performance Computing (HPC) and confront larger problem sizes, the choice becomes increasingly stark: to compromise data fidelity or to reduce resolution. To navigate this challenge, it is critical to consider in situ analysis and visualization techniques. These allow more frequent data "snapshots" to be taken directly from memory, thus avoiding the need for disruptive checkpointing.

In collaboration with Argonne scientist, Victor A. Mateevitsi, we have integrated NekRS with SENSEI and Ascent infrastructure to support *in situ* visualization capabilities. For SENSEI, a PR (pull request) is available at `https://github.com/Nek5000/nekRS/pull/528` with a related paper titled "Scaling Computational Fluid Dynamics: In Situ Visualization of NekRS using SENSEI" [70] which will be presented at the ISAV 2023 workshop of SC23 in November 2023. These enhancements aim to allow for scalable and efficient visualization of CFD simulations directly within NekRS and reduce post-processing efforts and storage requirements. In [70], tests were conducted with 146 pebbles for 3000 timesteps with 100 timestep intervals running on 70 nodes (280 ranks), 140 nodes (560 ranks), and 280 nodes (1120 ranks) on ALCF Polaris. The storage demand was a mere 6.5MB in contrast to 19GB necessitated by conventional checkpointing.

Figure 49 demonstrates preliminary experiment performed on Frontier using 1 node (8 GCDs) for 146 pebbles ($E = 62132$, $N = 9$, and the total number of gridpoints $n = 45$M; 5.M gridpoints per GCD) using Ascent (version 0.80). Two configurations with clip and slice were used defining contour on pebble at velocity magnitude $= 0.4$, with clip radius < 0.6 and call Ascent's function every 20 timesteps, and directly render on device and only save images. The output size per IO-step by default checkpointing is 1.1GB, compared to the Ascent image 4MB. The PR for the integration of NekRS with Ascent will be soon available as a next step.

8. OTHER PROJECT ACTIVITIES

8.1 2023 Gordon Bell Finalist

Several CEED members, Paul Fischer, Misun Min, Stefan Kerkemeier, Yu-Hsiang Lan, Malachi Phillips, Thilina Rathnayake, Noel Chalmers, and Tim Warburton, are 2023 Gordon Bell Finalist for their contributions to "Exascale Multiphysics Nuclear Reactor Simulations for Advanced Designs."

8.2 2023 R&D100 Award Winner

Two CEED members, Misun Min and Paul Fischer, are 2023 R&D 100 Award Winners for their contributions to "Cardinal: Accelerating Discovery in Fusion and Fission Energy" with ANL and INL nuclear engineers.

8.3 Seventh CEED Annual Meeting

The CEED project held its seventh annual meeting August 1-3, 2023 in a hybrid format: in-person at Lawrence Livermore National Laboratory and virtually using ECP Zoom for videoconferencing and Slack for side discussions with participation from 112 researchers (70 in-person) from 38 different organizations. The goal of the meeting was to report on the progress in the center, deepen existing and establish new connections with ECP hardware vendors, ECP software technologies projects and other collaborators, plan project activities and brainstorm/work as a group to make technical progress. In addition to gathering together many of the CEED researchers, the meeting included representatives of the ECP management, hardware vendors, software technology and other interested projects. See the meeting page at `https://ceed.exascaleproject.org/ceed7am` and this summary article `https://computing.llnl.gov/about/newsroom/ecp-ceed-2` for additional information.

8.4 SIAM-CSE, ParCFD, ICOSAHOM and ATPESC Participation

The CEED team organized two minisymposia at the SIAM Conference on Computational Science and Engineering (CSE23), inviting 16 speakers from various institutions in US and Europe. We also participated in the numerical libraries day of ATPESC and organized a minisimposium at the International Conference on Spectral and High Order Methods, where CEED had two members on the organizing committee and Misun Min presented one of the plenary talks. A CEED member, Misun Min, delivered the opening keynote talk, titled CFD at Exascale, at ParCFD 2023, held in Cuenca, Ecuador.

8.5 MFEM Tutorial on AWS

The MFEM team held its second cloud computing tutorial as part of LLNL's RADIUSS AWS tutorial series. The tutorial provided a self-paced overview of MFEM and its use for scalable finite element discretizations and application development. More than 40 participants followed the web-based lessons in their own Amazon EC2 instances. See the tutorial page at `https://mfem.org/tutorial` for additional information.

8.6 MAGMA Paper at SC23

The MAGMA team will participate and represent the CEED project at SC'23. A paper on GPU-based LU Factorization and Solve on Batches of Matrices with Band Structure [3] is accepted and will be presented in the 14th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Heterogeneous Systems (ScalAH'23).

8.7 ANL-BSC Collaboration for Turbulence Modeling

The Nek team at Argonne collaborates with a team at Barcelona Supercomputing Center (BSC) for turbulence modeling, which was extended from the development work that came out from the CEED–ExaWind collaboration. The research outcome between ANL and BSC has been reported in [62].

9. CONCLUSION

The goal of this milestone was to document and popularize the CEED-developed software and standards as part of the completion of the CEED efforts.

In addition to a final report on the CEED work for Frontier, Aurora, and ECP applications, this milestone included developments to engage external applications in the DOE and industry, the public release of the CEED-6.0 software distribution and the next CEED Annual meeting (CEED7AM) which included representatives from ECP applications, vendors and software technology projects.

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-855149.

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] BLT: A streamlined CMake build system foundation for developing HPC software. GitHub. `https://github.com/llNL/blt`.

[2] Parallel Adapt Results GitHub repository.

[3] Ahmad Abdelfattah, Stanimire Tomov, Piotr Luszczek, Hartwig Anzt, and Jack Dongarra. GPU-based LU Factorization and Solve on Batches of Matrices with Band Structure. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 14th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Heterogeneous Systems (ScalAH'23), to appear*, SC '23. Association for Computing Machinery, 2023.

[4] R. W. Anderson, V. A. Dobrev, T. V. Kolev, D. Kuzmin, M. Quezada de Luna, R. N. Rieben, and V. Z. Tomov. High-order local maximum principle preserving (MPP) discontinuous Galerkin finite element method for the transport equation. *J. Comput. Phys.*, 334:102–124, 2017.

[5] R. W. Anderson, V. A. Dobrev, T. V. Kolev, and R. N. Rieben. Monotonicity in high-order curvilinear finite element arbitrary Lagrangian–Eulerian remap. *Internat. J. Numer. Methods Engrg.*, 77(5):249–273, 2015.

[6] 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.

[7] Robert Anderson, Julian Andrej, Andrew Barker, Jamie Bramwell, Jean-Sylvain Camier, Jakub Cerveny, Veselin A. Dobrev, Yohann Dudouit, Aaron Fisher, Tzanio V. Kolev, Will Pazner, Mark Stowell, Vladimir Z. Tomov, Ido Akkerman, Johann Dahm, David Medina, and Stefano Zampini. MFEM: a modular finite elements methods library. *Comput. Math. Appl.*, 81:42–74, 2021.

[8] Mario Arioli. A stopping criterion for the conjugate gradient algorithm in a finite element method framework. *Numer. Math.*, 97:1–24, 2004.

[9] Mario Arioli, Emmanuil H Georgoulis, and Daniel Loghin. Stopping criteria for adaptive finite element solvers. *SIAM J. Sci. Comput.*, 35(3):A1537–A1559, 2013.

[10] Mario Arioli, Jörg Liesen, Agnieszka Międlar, and Zdeněk Strakoš. Interplay between discretization and algebraic computation in adaptive numerical solution of elliptic PDE problems. *GAMM Mitt.*, 36(1):102–129, 2013.

[11] Mario Arioli, Daniel Loghin, and Andy J Wathen. Stopping criteria for iterations in finite element methods. *Numer. Math.*, 99:381–410, 2005.

[12] Anthony P Austin, Noel Chalmers, and Tim Warburton. Initial guesses for sequences of linear systems in a gpu-accelerated incompressible flow solver. *SIAM Journal on Scientific Computing*, 43(4):C259–C289, 2021.

[13] V. Balasubramanian, A. Treikalis, O. Weidner, and S. Jha. Ensemble toolkit: Scalable and flexible execution of ensembles of tasks. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 458–463, 2016.

[14] R.J. Beare and et. al. An intercomparison of large-eddy simulations of the stable boundary layer. *Boundary-Layer Meteorology*, (118):247–272, 2006.

[15] Robert J Beare, Malcolm K Macvean, Albert AM Holtslag, Joan Cuxart, Igor Esau, Jean-Christophe Golaz, Maria A Jimenez, Marat Khairoutdinov, Branko Kosovic, David Lewellen, et al. An intercomparison of large-eddy simulations of the stable boundary layer. *Boundary-Layer Meteorology*, 118(2):247–272, 2006.

[16] D. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. Kunen, O. Pearce, P. Robinson, B. Ryujin, and T. Scogland. RAJA: Portable performance for large-scale scientific applications. In *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 71–81. IEEE, 2019.

[17] D. A. Beckingsale, M. J. McFadden, J. P. S. Dahm, R. Pankajakshan, and R. D. Hornung. Umpire: Application-focused management and coordination of complex hierarchical memory. *IBM Journal of Research and Development*, 64(3/4):00:1–00:10, 2020.

[18] David Beckingsale, Marty McFadden, Johann Dahm, Ramesh Pankajakshan, and Rich Hornung. Umpire: Application-focused management and coordination of complex hierarchical memory, May 2020.

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

[20] Pedro D. Bello-Maldonado, Tzanio V. Kolev, Robert N. Rieben, and Vladimir Z. Tomov. A matrix-free hyperviscosity formulation for high-order ALE hydrodynamics. *Comput. Fluids*, 205:104577, 2020.

[21] Christine Bernardi and Rüdiger Verfurth. Adaptive finite element methods for elliptic equations with non-smooth coefficients. *Numer. Math.*, 85(4):579–608, 2000.

[22] Jed Brown, Ahmad Abdelfatah, Jean-Sylvain Camier, Veselin Dobrev, Jack Dongarra, Paul Fischer, Aaron Fisher, Yohann Dudouit, Azzam Haidar, Kazem Kamran, Tzanio Kolev, Misun Min, Thilina Ratnayaka, Mark Shephard, Cameron Smith, Stanimire Tomov, Vladimir Tomov, and Tim Warburton. ECP Milestone Report CEED-MS13: Public release of CEED 1.0, April 2, 2018.

[23] Jean-Sylvain Camier, Veselin Dobrev, Patrick Knupp, Tzanio Kolev, Ketan Mittal, Robert Rieben, and Vladimir Z. Tomov. Accelerating high-order mesh optimization using finite element partial assembly on GPUs. *J. Comput. Phys.*, 474:111808, 2023.

[24] Robert Carson, John Coleman, and Matt Rolchigo. Exaam uq workflow, 2023.

[25] N Chalmers, A Karakus, AP Austin, K Swirydowicz, and T Warburton. libparanumal: a performance portable high-order finite element library, 2020. *Release 0.4. 0.*

[26] Noel Chalmers, Jakub Kurzak, Damon McDougall, and Paul T Bauman. Optimizing high-performance linpack for exascale accelerated architectures. *arXiv preprint arXiv:2304.10397*, 2023.

[27] Noel Chalmers, Abhishek Mishra, Damon McDougall, and Tim Warburton. Hipbone: A performance-portable graphics processing unit-accelerated c++ version of the nekbone benchma rk. *The International Journal of High Performance Computing Applications*, 37(5):560–577, 2023.

[28] Noel Chalmers and Tim Warburton. Portable high-order finite element kernels I: Streaming operations. *arXiv preprint arXiv:2009.10917*, 2020.

[29] Noel Chalmers and Timothy Warburton. Low-order preconditioning of high-order triangular finite elements. *SIAM Journal on Scientific Computing*, 40(6):A4040–A4059, 2018.

[30] Sunita Chandrasekaran and Guido Juckeland. *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley Professional, 2017.

[31] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):1–14, 2008.

[32] Stephen Cleary and Paul A. Bristow. Boost.pool.

[33] M.O. Deville, P.F. Fischer, and E.H. Mund. *High-order methods for incompressible fluid flow*. Cambridge University Press, Cambridge, 2002.

[34] Wei Ding, Ligang Lu, Mauricio Araya-Polo, Amik St-Cyr, Detlef Hohl, and Barbara M Chapman. Using gpu shared memory with a directive-based approach. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 1021–1028. IEEE, 2014.

[35] V. Dobrev, Tz. Kolev, and R. Rieben. High-order curvilinear finite element methods for Lagrangian hydrodynamics. *SIAM J. Sci. Comp.*, 34(5):606–641, 2012.

[36] 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.

[37] Veselin A. Dobrev, Patrick Knupp, Tzanio V. Kolev, Ketan Mittal, Robert N. Rieben, and Vladimir Z. Tomov. Simulation-driven optimization of high-order meshes in ALE hydrodynamics. *Comput. Fluids*, 2020.

[38] Veselin A. Dobrev, Patrick Knupp, Tzanio V. Kolev, Ketan Mittal, and Vladimir Z. Tomov. The Target-Matrix Optimization Paradigm for high-order meshes. *SIAM J. Sci. Comp.*, 41(1):B50–B68, 2019.

[39] H Carter Edwards, Christian R Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of parallel and distributed computing*, 74(12):3202–3216, 2014.

[40] Alexandre Ern and Martin Vohralík. Adaptive inexact Newton methods with a posteriori stopping criteria for nonlinear diffusion PDEs. *SIAM J. Sci. Comput.*, 35(4):A1761–A1791, 2013.

[41] 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.

[42] P. Fischer, S. Kerkemeier, M. Min, Y.H. Lan, M. Phillips, T. Rathnayake, E. Merzari, A. Tomboulides, A. Karakus, N. Chalmers, and T. Warburton. NekRS, a GPU-accelerated spectral element Navier-Stokes solver. *CoRR*, abs/2104.05829, 2021.

[43] Paul Fischer, Stefan Kerkemeier, Misun Min, Yu-Hsiang Lan, Malachi Phillips, Thilina Rathnayake, Elia Merzari, Ananias Tomboulides, Ali Karakus, Noel Chalmers, et al. Nekrs, a gpu-accelerated spectral element navier–stokes solver. *Parallel Computing*, 114:102982, 2022.

[44] P.F. Fischer. Parallel multi-level solvers for spectral element methods. In A.V. Ilin and L.R. Scott, editors, *Third Int. Conference on Spectral and High Order Methods*, pages 595–604. Houston J. of Mathematics, 1996.

[45] H. Grotjans and F. Menter. Wall functions for general application cfd codes. In *ECCOMAS 98, Proceedings of the 4th European Computational Fluid Dynamics Conference, John Wiley & Sons*, pages 1112–1112, 1998.

[46] Khronos SYCL Working Group et al. Sycl 2020 specification, 2021.

[47] Yichen Guo, Eric de Sturler, and Tim Warburton. Stopping criteria for the conjugate gradient algorithm in high-order finite element methods. *arXiv preprint arXiv:2305.10965*, 2023.

[48] Yichen Guo, Paul Fischer, and Misun Min. High order methods for incompressible magnetohydrodynamics. ANL technical report, 2023.

[49] Hennes Hajduk, Dmitri Kuzmin, Tzanio V. Kolev, and Remi Abgrall. Matrix-free subcell residual distribution for Bernstein finite element discretizations of linear advection equations. *Comput. Methods Appl. Mech. Eng.*, 359, 2020.

[50] Hennes Hajduk, Dmitri Kuzmin, Tzanio V. Kolev, Vladimir Z. Tomov, Ignacio Tomas, and John N. Shadid. Matrix-free subcell residual distribution for Bernstein finite elements: Monolithic limiting. *Comput. Fluids*, 200:104451, 2020.

[51] Christopher Hauer, Elmar Nöth, Alexander Barnhill, Andreas Maier, Julius Guthunz, Heribert Hofer, Rachael Xi Cheng, Volker Barth, and Christian Bergler. Orca-spy enables killer whale sound source simulation, detection, classification and localization using an integrated deep learning-based segmentation. *Scientific Reports*, 13(1):11106, 2023.

[52] VE Henson and U.M. Yang. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, 2002.

[53] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications.* Springer Science & Business Media, 2007.

[54] Ralf Hiptmair. Finite elements in computational electromagnetism. *Acta Numerica*, 11:237–339, 2002.

[55] Daniel Alejandro Ibanez. *Conformal mesh adaptation on heterogeneous supercomputers.* Rensselaer Polytechnic Institute, Troy, NY, 2016.

[56] Pavel Jiránek, Zdeněk Strakoš, and Martin Vohralík. A posteriori error estimates including algebraic error and stopping criteria for iterative solvers. *SIAM J. Sci. Comput.*, 32(3):1567–1590, 2010.

[57] Ali Karakus, Noel Chalmers, Jan S Hesthaven, and Tim Warburton. Discontinuous galerkin discretizations of the boltzmann–bgk equations for nearly incompressible flows: Semi-analytic time stepping and absorbing boundary layers. *Journal of Computational Physics*, 390:175–202, 2019.

[58] Ali Karakus, Noel Chalmers, and Tim Warburton. A local discontinuous galerkin level set reinitialization with subcell stabilization on unstructured meshes. *Computers & Mathematics with Applications*, 123:160–170, 2022.

[59] 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.

[60] Tzanio Kolev, Paul Fischer, Ahmad Abdelfattah, Natalie Beams, Jed Brown, Jean-Sylvain Camier, Robert Carson, Noel Chalmers, Veselin Dobrev, Yohann Dudouit, Leila Ghaffari, Aditya Y. Joshi, Stefan Kerkemeier, Yu-Hsiang Lan, Damon McDougall, David Medina, Misun Min, Abhishek Mishra, Will Pazner, Malachi Phillips, Thilina Ratnayaka, Mark S. Shephard, Morteza H. Siboni, Cameron W. Smith, Jeremy L. Thompson, Ananias Tomboulides, Stanimire Tomov, Vladimir Tomov, and Tim Warburton. ECP Milestone Report CEED-MS38: High-order algorithmic developments and optimizations for more robust exascale applications, March 31, 2022.

[61] Tzanio V Kolev and Panayot S Vassilevski. Parallel auxiliary space AMG solver for H(div) problems. *SIAM Journal on Scientific Computing*, 34(6):A3079–A3098, 2012.

[62] Vishal Kumar, Oriol Lehmkuhl, Ananias Tombouldies, Paul Fischer, and Misun Min. Turbulence modeling with nek5000/rs, sod2d and alya. Technical report, Argonne National Laboratory, September 2023. ANL-23/59.

[63] D. Kuzmin, O. Mierka, and S. Turek. On the implementation of the Îº-Îµ turbulence model in incompressible flow solvers based on a finite element discretisation. *International Journal of Computing Science and Mathematics*, 1(2-4):193–206, 2007.

[64] Laghos: High-order Lagrangian hydrodynamics miniapp, 2023. `http://github.com/CEED/Laghos`.

[65] Yu-Hsiang Lan, Paul Fischer, Elia Merzari, and Misun Min. All-Hex Meshing Strategies For Densely Packed Spheres. *Proceedings of the 29th International Meshing Roundtable*, pages 293–305, June 2021.

[66] Neil Lindquist, Paul Fischer, and Misun Min. Scalable interpolation on gpus for thermal fluids applications. Technical Report ANL-21/55, Argonne National Laboratory, 2021.

[67] Gabriel H Loh, Michael J Schulte, Mike Ignatowski, Vignesh Adhinarayanan, Shaizeen Aga, Derrick Aguren, Varun Agrawal, Ashwin M Aji, Johnathan Alsop, Paul Bauman, et al. A research retrospective on amd's exascale computing journey. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023.

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

[69] Mathias Louboutin, Michael Lange, Fabio Luporini, Navjot Kukreja, Philipp A Witte, Felix J Herrmann, Paulius Velesko, and Gerard J Gorman. Devito (v3. 1.0): an embedded domain-specific language for finite differences and geophysical exploration. *Geoscientific Model Development*, 12(3):1165–1187, 2019.

[70] Victor A. Mateevitsi, Mathis Bode, Nicola Ferrier, Paul Fischer, Jens Henrik Göbbert, Joseph A. Insley, Yu-Hsiang Lan, Misun Min, Michael E. Papka, Saumil Patel, Silvio Rizzi, and Jonathan Windgassen. Scaling computational fluid dynamics: In situ visualization of nekrs using sensei. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC23)*, 2023. accepted.

[71] David S Medina, Amik St-Cyr, and Tim Warburton. Occa: A unified approach to multi-threading languages. *arXiv preprint arXiv:1403.0968*, 2014.

[72] Elia Merzaria, Steven Hamilton, Thomas Evans, Paul Romano, Paul Fischer, Misun Min, Stefan Kerkemeier, Yu-Hsiang Lan, Jun Fang, Malachi Phillips, Thilina Rathnayake, Elliott Biondo, Katherine Royston, Noel Chalmers, and Tim Warburton. Exascale multiphysics nuclear reactor simulations for advanced designs. *2023 International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2023. accepted.

[73] M. Min, Y. Lan, P. Fischer, E. Merzari, S. Kerkemeier, M. Phillips, T. Rathnayake, A. Novak, D. Gaston, N. Chalmers, and T. Warburton. Optimization of full-core reactor simulations on Summit. In *Proc. Conf. on Supercomput.* IEEE, 2022.

[74] Misun Min, Michael Brazell, Ananias Tomboulides, Matthew Churchfield, Paul Fischer, and Michael Sprague. Towards exascale for wind enery simulations. revision, 2023.

[75] 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.

[76] Misun Min, Yu-Hsiang Lan, P. Fischer, E. Merzari, S. Kerkemeier, , M. Phillips, T. Rathnayake, A. Novak, D. Gaston, N. Chalmers, and T. Warburton. Optimization of full-core reactor simulations on summit. *2022 International Conference for High Performance Computing, Networking, Storage, and Analysis*, pages 1–11, 2022.

[77] Misun Min, Yu-Hsiang Lan, Paul Fischer, Thilina Rathnayake, and John Holmen. Nek5000/rs performance on advanced gpu architectures. *Frontiers in High Performance Computing*, submitted 2023.

[78] Misun Min and Ananias Tomboulides. Simulating atmospheric boundary layer turbulence with Nek5000/RS. Technical Report ANL-22/79, Argonne National Laboratory, 2022.

[79] Ketan Mittal. *Highly scalable solution of incompressible Navier-Stokes equations using the spectral element method with overlapping grids.* PhD thesis, University of Illinois Urbana-Champaign, 2019.

[80] Ketan Mittal, Som Dutta, and Paul Fischer. Nonconforming Schwarz-spectral element methods for incompressible flow. *Computers and Fluids*, 191, 2019.

[81] Ketan Mittal, Som Dutta, and Paul Fischer. Multirate time-stepping for the incompressible Navier-Stokes equations in overlapping grids. 437:110335, 2020.

[82] A. S. Monin and A. M. Obukhov. Basic laws of turbulent mixing in the surface layer of the atmosphere. *Tr. Akad. Nauk. SSSR Geophiz. Inst.*, 24(151):163—-187, 1954.

[83] WA Mulder and R-E Plessix. Exploring some issues in acoustic full waveform inversion. *Geophysical Prospecting*, 56(6):827–841, 2008.

[84] Aaftab Munshi, Benedict Gaster, Timothy G Mattson, and Dan Ginsburg. *OpenCL programming guide.* Pearson Education, 2011.

[85] Bradford Nichols, Dick Buttlar, and Jacqueline Farrell. *Pthreads programming: A POSIX standard for better multiprocessing.* " O'Reilly Media, Inc.", 1996.

[86] Jan Papež, Ulrich Rüde, Martin Vohralík, and Barbara Wohlmuth. Sharp algebraic and total a posteriori error bounds for h and p finite elements via a multilevel approach. Recovering mass balance in any situation. *Comput. Methods Appl. Mech. Eng.*, 371:113243, 2020.

[87] Aditya Parik, Paul Fischer, and Misun Min. High performance lagrangian particle tracking. ANL technical report, 2023.

[88] parRSB Development Team. parrsb: Parallel recursive spectal bisection. `https://github.com/Nek5000/parRSB.git`, 2023.

[89] A.T. Patera. A spectral element method for fluid dynamics : laminar flow in a channel expansion. 54:468–488, 1984.

[90] Will Pazner, Tzanio Kolev, and Panayot Vassilevski. Matrix-free GPU-accelerated saddle-point solvers for high-order problems in H(div), 2023.

[91] M. Phillips, S. Kerkemeier, and P. Fischer. *Tuning Spectral Element Preconditioners for Parallel Scalability on GPUs*, pages 37–48. 2022.

[92] M. Phillips, S. Kerkemeier, and P. Fischer. Tuning spectral element preconditioners for parallel scalability on GPUs. In *Proc. of the 2022 SIAM Conf. on Par. Proc. for Sci. Comp.*, pages 37–48. SIAM, 2022.

[93] Malachi Phillips and Paul Fischer. Optimal chebyshev smoothers and one-sided V-cycles. *arXiv preprint arXiv:2210.03179*, 2022.

[94] M Picasso. A stopping criterion for the conjugate gradient algorithm in the framework of anisotropic adaptive finite elements. *Commun. Numer. Methods Eng.*, 25(4):339–355, 2009.

[95] Finnur Pind, Cheol-Ho Jeong, Allan P Engsig-Karup, Jan S Hesthaven, and Jakob Strømann-Andersen. Time-domain room acoustic simulations with extended-reacting porous absorbers using the discontinuous galerkin method. *The Journal of the Acoustical Society of America*, 148(5):2851–2863, 2020.

[96] Ahmad Qawasmeh, Maxime R Hugues, Henri Calandra, and Barbara M Chapman. Performance portability in reverse time migration and seismic modelling via openacc. *The International Journal of High Performance Computing Applications*, 31(5):422–440, 2017.

[97] Remhos: Remap high-order solver, 2023. `http://github.com/CEED/Remhos`.

[98] Javier Rodrigo, Matthew Churchfield, and Brank Kosovic. A methodology for the design and testing of atmospheric boundary layer models for wind energy applications. *Wind Energ. Sci.*, 2:23–54, 2017.

[99] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[100] U. Schumann. Subgrid scale model for finite difference simulations of turbulent flows in plane channels and annuli. *Journal of Computational Physics*, 18(4):376–404, 1975.

[101] U. Schumann. Subgrid scale model for finite difference simulations of turbulent flows in plane channels and annuli. *Journal of Computational Physics*, (18):376–404, 1975.

[102] S Shiraiwa, N Bertelli, W Tierens, R Bilato, J Hillairet, J Myra, H Kohno, M Poulos, and M Ono. Magnetic potential based formulation for linear and non-linear 3d rf sheath simulation. *Nuclear Fusion*, 63(2):026024, 2023.

[103] 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.

[104] 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.

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

[106] Morteza H Siboni and Mark S Shephard. Adaptive workflow for simulation of rf heaters. *Computer Physics Communications*, 279:108434, 2022.

[107] J.S. Smagorinsky. General circulation experiments with the primitive equations. *Mon. Weather Rev.*, 91:99–164, 1963.

[108] A St-Cyr, S Reker, and JW Blokland. Large scale full waveform induced seismicity inversion for groningen. In *First EAGE Workshop on High Performance Computing for Upstream in Latin America*, volume 2018, pages 1–5. European Association of Geoscientists & Engineers, 2018.

[109] A St-Cyr, S Reker, S Frijters, S Chawdhary, A Panda, S Banerjee, H Knibbe, and M Muruganantham. Gpu accelerated fwi using the open concurrent computing abstraction (occa). In *Fifth EAGE Workshop on High Performance Computing for Upstream*, volume 2021, pages 1–5. European Association of Geoscientists & Engineers, 2021.

[110] Thomas Stitt, Kristi Belcher, Alejandro Campos, Tzanio Kolev, Philip Mocz, Robert N. Rieben, Aaron Skinner, Vladimir Z. Tomov, Aarturo Vargas, and Kenneth Weiss. Performance portable GPU acceleration of a high-order finite element multiphysics application. *submitted at J. Fluids Eng.*, 2023.

[111] S. Stolz, P. Schlatter, and L. Kleiser. High-pass filtered eddy-viscosity models for large-eddy simulations of transitional and turbulent flow. *Physics of Fluids*, 17(6):065103, 2005.

[112] P. Sullivan, J. McWilliams, and C.H. Moeng. A subgrid-scale model for large-eddy simulation of planetary boundary-layer flows. *Bound.-Layer Meteor.*, 71:247–276, 1994.

[113] Kasia Świrydowicz, Noel Chalmers, Ali Karakus, and Tim Warburton. Acceleration of tensor-product operations for high-order finite element methods. *The International Journal of High Performance Computing Applications*, 33(4):735–757, 2019.

[114] Zhenyu Tang, Rohith Aralikatti, Anton Jeran Ratnarajah, and Dinesh Manocha. Gwa: A large high-quality acoustic dataset for audio processing. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022.

[115] Ananais Tomboulides, Misun Min, Paul Fischer, Matt Churchfield, and Michael Sprague. Advanced turbulence models for large-scale atmospheric boundary layer flows. Technical Report ANL-23/60, Argonne National Laboratory, 2023.

[116] Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):805–817, 2022.

[117] H.M. Tufo and P.F. Fischer. Fast parallel direct solvers for coarse grid problems. *J. Parallel Distrib. Comput.*, 61:151–177, 2001.

[118] Ruud Van der Pas, Eric Stotzer, and Christian Terboven. *Using OpenMP# the next step: affinity, accelerators, tasking, and simd.* MIT press, 2017.

[119] Arturo Vargas, Thomas M. Stitt, Kenneth Weiss, Vladimir Z. Tomov, Jean-Sylvain Camier, Tzanio Kolev, and Robert N. Rieben. Matrix-free approaches for GPU acceleration of a high-order finite element hydrodynamics application using MFEM, Umpire, and RAJA. *Int. J. High Perform. Comput. Appl.*, 36(4):492–509, 2022.

[120] Jean Virieux. P-sv wave propagation in heterogeneous media: Velocity-stress finite-difference method. *Geophysics*, 51(4):889–901, 1986.

[121] Erik Zenker, Benjamin Worpitz, René Widera, Axel Huebl, Guido Juckeland, Andreas Knüpfer, Wolfgang E Nagel, and Michael Bussmann. Alpaka–an abstraction library for parallel kernel acceleration. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 631–640. IEEE, 2016.