ECP Milestone Report

Improve performance and capabilities of CEED-enabled ECP applications on Frontier/Aurora EA

WBS 2.2.6.06, Milestone CEED-MS39

Tzanio Kolev
Paul Fischer
Ahmad Abdelfattah
Adeleke Bankole
Natalie Beams
Michael Brazell
Jed Brown
Jean-Sylvain Camier
Noel Chalmers
Matthew Church eld
Veselin Dobrev
Yohann Dudouit
Leila Gha ari
John Holmen
Stefan Kerkemeier
Yu-Hsiang Lan
Yimin Lin
Damon McDougall
Elia Merzari
Misun Min
Ketan Mittal
Will Pazner
Malachi Phillips
Thilina Ratnayaka
Kris Rowe
Mark S. Shephard
Cameron W. Smith
Michael Sprague
Jeremy L. Thompson
Ananias Tomboulides
Stanimire Tomov
Vladimir Tomov
Tim Warburton
James Wright III

September 30, 2022

ECP Milestone Report
Improve performance and capabilities of CEED-enabled ECP applications on Frontier/Aurora EA
WBS 2.2.6.06, Milestone CEED-MS39

O ce of Advanced Scienti c Computing Research
O ce of Science
US Department of Energy

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

September 30, 2022

ECP Milestone Report
Improve performance and capabilities of CEED-enabled ECP applications on Frontier/Aurora EA
WBS 2.2.6.06, Milestone CEED-MS39

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	October 5, 2022	Original	

EXECUTIVE SUMMARY

The goal of this milestone was to port the CEED software stack, including Nek, MFEM and libCEED to Frontier and/or Aurora early access hardware, work on optimizing the performance on AMD and Intel GPUs, and demonstrate impact in CEED-enabled ECP applications.

As part of this milestone, we also organized the next CEED annual meeting (CEED6AM) which included representatives from ECP applications, vendors and software technology projects, hosted the Nek user meeting and MFEM AWS tutorial, and worked on a number of additional software and algorithmic improvements.

The speci c tasks addressed in this milestone were as follows.

- CEED-T21 (ADCD04-91): Port full CEED software stack to Frontier/Aurora EA, report benchmark results.
- CEED-T22 (ADCD04-92): Help in application porting to Frontier/Aurora EA, develop application-speci c optimizations.
- CEED-T23 (ADCD04-93): Multi-node scaling on Frontier (strong and weak scaling).
- CEED-T24 (ADCD04-94): CEED Annual meeting (CEED6AM).

TABLE OF CONTENTS

Executive Summary **vi**

List of Figures **viii**

List of Tables **x**

1 Introduction **1**

2 CEED Performance Improvements for Frontier **1**
- 2.1 NekRS performance on Summit, Perlmutter, Polaris, Crusher and Frontier for ExaSMR . . . 1
- 2.2 AMR-Wind and NekRS performance on Summit and Crusher for ExaWind 5
- 2.3 MFEM fused GPU solvers for the DG mass operator in Laghos 11
- 2.4 CEED benchmarks with matrix instructions on AMD MI250X 13
- 2.5 Performance tuning for the non-tensor MAGMA backend on AMD GPUs 17
- 2.6 Solid mechanics with matrix-free p-multigrid on Lassen, Summit, Perlmutter and Crusher . . 18
- 2.7 Mixed-precision libCEED kernels and compiler insights for AMD MI250X 23

3 CEED Performance Improvements for Aurora **25**
- 3.1 Extending the MAGMA portability with OneAPI 25
- 3.2 NekRS on Ponte Vecchio with Intel OneAPI DPC++ implementation 28

4 Additional Topics **28**
- 4.1 Partial assembly for high-order mesh optimization in MFEM 29
- 4.2 Solver advances for high-order discretizations 30
- 4.3 Algorithmic tradeo s for implicit uids solvers 31
- 4.4 Performance tuning for the non-tensor MAGMA backend on NVIDIA GPUs 32
- 4.5 All-hex meshing for packed beds with contacting spheres 32
- 4.6 Spectral elements for coupled PNP-NS solver 33
- 4.7 Exascale simulation work ow 34

5 Other Project Activities **34**
- 5.1 Sixth CEED annual meeting 35
- 5.2 Nek user meeting 35
- 5.3 SIAM-CSE 2023 and ICOSAHOM 2023 participation 35
- 5.4 ALCF GPU hackathon on Polaris 35
- 5.5 MFEM tutorial on AWS 35
- 5.6 NekRS contribution to Cardinal, 2022 R&D100 nalist 36
- 5.7 2023 INCITE submission 36

6 Conclusion **36**

LIST OF FIGURES

1 Strong-scaling on Crusher, Summit, Perlmutter and Polaris (17×17 rod bundle with 10 layers). 2
2 Strong-scaling on Crusher, Summit, Perlmutter and Polaris (17×17 rod bundle with 170 layers). 3
3 Strong-scaling on Frontier (cray-mpich/8.1.17, rocm/5.1.0) and Crusher (cray-mpich/8.1.19, rocm/5.2.0) for rod17 × 17 with 170 layers ($E = 27700 \times 170$). **Upper Left:** time-per-step vs. n P for Frontier and Crusher **Upper Right:** Comparative time-per-step vs. n P for Crusher (MI250X), Summit (V100), Perlmutter (A100), and Polaris (A100). **Lower Left:** Time in the work-intensive advection operator for Frontier and Crusher. **Lower Right:** Time in the communication-intensive coarse-grid solve for Frontier and Crusher. 4
4 NekRS vs AMR-Wind GPU cost breakdown on Summit (top) and Crusher (bottom), using $n = 512^3$ and 2000 steps. 8
5 NekRS and AMR-Wind: CPU vs. GPU performance on Summit: 100 steps average from 200 step runs for $n = 512^3$. 9
6 NekRS vs. AMR-Wind strong and weak scaling on Summit GPUs. 9
7 NekRS GPU strong-scaling comparison on Crusher and Summit. 12
8 **Comparison of the setup throughput for the different solvers.** The setup throughput is the same for all conjugate gradient based solvers, either local or global; on the other hand direct solvers have a much higher cost. Direct solvers should therefore be avoided in favor of conjugate gradient solvers when the mass matrix changes over time. 13
9 **Comparison of the total solve throughput for the different solvers.** The global conjugate gradient peak performance is around 20-30 MDOF/s, which is about two orders of magnitude slower than the best performing solver. Local conjugate gradient solvers achieve good performance for the full range of polynomial orders. Direct solvers constitute an interesting alternative, especially explicit inverse for low orders. 14
10 Throughput for three di erent implementations of the BS9 asymmetric streaming benchmark executing on a single GCD of an AMD MI250X using ROCm 5.2.0. 16
11 Comparison of the NVIDIA GA100 streaming multiprocessor and AMD MI200 series CDNA2 core architectures. 16
12 Shape of matrix multiplication for the non-tensor basis actions 18
13 Colormap of the DGEMM selector for the MI250X 18
14 Performance of the 3D di usion problem using MAGMA's non-tensor backend on the MI100 GPU. 19
15 Performance of the 3D di usion problem using MAGMA's non-tensor backend on the MI250X GPU 19
16 Visualization of the deformed state and strain energy singularities on `mesh A` of Figure 17, re ned 3 times by splitting each hexahedron in 8 without snapping to geometry, and solved using Q_2 nite elements. There are physical singularities on the back surface (e.g., top-right corner) and non-physical singularities at the weak reentrant corners of the hole (which do not exist in the smooth model with exact cylinder). 20
17 Accuracy study showing relative error in total strain energy versus DoFs for the bending experiment Figure 16 under both h re nement (same shape) and p re nement (same color) with low and high order geometry. The Pareto front is toward the lower left and we observe that h re nement always moves away from optimality. The slope of h re nement is the same for all meshes and solution orders. p re nement is very e cient so long as the geometry is at least quadratic, but causes errors to increase when p re ning on linear geometry due to resolution of the non-physical singularities. 20
18 Extruded Schwarz Primitive surface under 12% compressive strain, colored by von Mises stress. The left wall is xed and a compressive force is applied to the facing surfaces on the right. The simulation used 2 re nements, 2 layers, thickness 0.2, and Q_2 elements. 21

19 Efficiency per Newton iteration versus time for Q_2 (**left**) and Q_3 (**right**) finite elements using matrix-free Newton-Krylov with p-MG preconditioning and BoomerAMG coarse solve. Problem sizes (in MDoF) are annotated for the minimum and maximum sizes for each host and number of nodes combination. The impact of latency is ever-present, with memory capacity limiting the right end of each curve. Ideal weak scaling is evident for Q_3 on Perlmutter for Newton step time above 1.8 s where the 1-node and 8-node curves coincide, while communication latency leads to degradation at the smallest problem sizes (2 MDoF/GPU with time around 1 s). . . 21
20 Linear solve efficiency spectrum for Q_2 (**left**) and Q_3 (**right**) finite elements using matrix-free Newton-Krylov with p-MG preconditioning and BoomerAMG coarse solve. The times and efficiencies are per Newton iteration. Problem sizes (in MDoF) are annotated for the minimum and maximum sizes for each host and number of nodes combination. 22
21 Preconditioner setup efficiency spectrum for Q_2 finite elements using matrix-free Newton-Krylov with p-MG preconditioning and BoomerAMG coarse solve. The times and efficiencies are per Newton iteration. Problem sizes (in MDoF) are annotated for the minimum and maximum sizes for each host and number of nodes combination. 22
22 Comparison of operators compiled with `O3` versus `O1` for the `magma-det` backend 23
23 Comparison of mixed-precision and single-precision performance on MI250X for different optimization flags in the `magma-det` backend 24
24 Comparison of data read/write as reported by rocprof for different precisions in `magma-det` backend 26
25 Left: Performance of MAGMA's SGEMM translated to DPC++ and benchmarked on multicore AMD EPYC 7742 64-Core 2.25 GHZ CPU. Right: Performance of MAGMA's SGEMM translated to DPC++ and benchmarked on NVIDIA GeForce RTX 3060 GPU. 27
26 Left: Performance MAGMA SGEMM versions translated to DPC++ and benchmarked on Intel UHD Graphics P630 GPU; Right: MAGMA's templated GEMM producing various versions (e.g., MAGMA ker2 and MAGMA ker11) through autotuning nine templated parameters. . . 27
27 Relative performance of NekRS benchmarks with problem size of 8196 (Averaged throughput, higher is better). 28
28 Kershaw meshes: (left) Initial mesh, (right) Mesh deformed using Kershaw deformation parameters $_y = {}_z = 0\;3$. 29
29 Throughput comparison - action of the second derivative operator. Single Lassen node throughput (in GDOF/s, i.e. billions of degrees of freedom per second) of (a) 40 IBM Power9 CPU cores and (b) 4 NVIDIA Tesla V100-SXM GPU. 30
30 Throughput of the complete TMOP computation. Single NVIDIA Tesla V100-SXM GPU throughput (in MDOF/s, i.e. millions of processed degrees of freedom per second). 31
31 Colormap of the DGEMM selector for the A100 33
32 Performance of the 3D diffusion problem using MAGMA's non-tensor backend on the V100 GPU. 33
33 Performance of the 3D diffusion problem using MAGMA's non-tensor backend on the A100 GPU 34

LIST OF TABLES

1 Problem setup for strong and weak scaling studies. 5
2 AMR-Wind performance optimization. 6
3 CUDA kernel statistics from NVIDIA® Nsight™ profiler using `nsys profile --stats=true -t nvtx,cuda`. 7
4 NekRS GPU vs. AMR-Wind GPU strong-scaling performance study. 10
5 NekRS GPU vs. AMR-Wind GPU weak-scaling performance study with fixed mesh density and resolution per GPU. 11
6 Two orders of magnitude Laghos speedup reached at order 4 with the fused GPU solver for the discontinuous Galerkin mass matrix. 13
7 Best obtained DEVICE memory throughputs and floating point throughputs for BK3 in double precision (FP64) and single precision (FP32) over a range of polynomial degrees (N). Calculations performed on a single GCD of an AMD MI250X GPU using HIP from ROCm 5.2.0. Results shown in red were obtained using a combination of regular vector and matrix instructions `__builtin_amdgcn_mfma_f64_4x4x4f64` whereas results shown in black were obtained using only regular vector instructions. $N + 2$ quadrature points were used in each direction for these experiments. 17
8 Selected HIP compiler output information for a gradient basis kernel from the MAGMA backend 25
9 For meshes of different orders (p), we compare (a) unique degrees of freedom and total quadrature points per core and (b) total time to solution. 30
10 Solver configuration with the fastest time to solution. 31
11 Execution time per preconditioner application and total solve time per time step for ceed-fluids on 2 nodes of Perlmutter. The new GPU implementation for variable point-block Jacobi is 5× faster than a mathematically equivalent preconditioner using cuSPARSE. 32

1. INTRODUCTION

In this milestone we ported the CEED software stack, including Nek, MFEM and libCEED to Frontier and/or Aurora early access hardware, worked on optimizing the performance on AMD and Intel GPUs, and demonstrated impact in CEED-enabled ECP applications. These activities are described in Section 2 and Section 3. As part of this milestone, we also organized the next CEED a meeting (CEED6AM) which included representatives from ECP applications, vendors and software technology projects, hosted the Nek user meeting and MFEM AWS tutorial, and worked on a number of additional software and algorithmic improvements. These activities are described in Section 4 and Section 5.

2. CEED PERFORMANCE IMPROVEMENTS FOR FRONTIER

2.1 NekRS performance on Summit, Perlmutter, Polaris, Crusher and Frontier for ExaSMR

We consider ExaSMR's 17×17 rod-bundle geometry and extend the domain in streamwise direction with 10, 17, 170 and 6340 layers keeping the mesh density same. In this report, we present the 10-layer and 170-layer cases, which correspond to 277 thousand spectral elements of order $N = 7$, for a total of $n =$ 27M $\times 7^3 =$ 95M grid points and 471 thousand spectral elements of order $N = 7$, for a total of $n =$ 47M $\times 7^3 =$ 161M grid points, respectively.

Figure 1 shows several scaling metrics for 10-layer case across the platforms of Crusher, Summit, Perlmutter and Polaris. In the top row, left, we have the classic strong-scaling plot of time-per-step (t_{step}), in seconds, versus number of MPI ranks, P. (We run one MPI rank per V100 or A100 on the NVIDIA-based nodes, and one MPI rank per GCD on the AMD MI250X nodes.) For early run results (shown here) on Crusher, there was a 2× slow down if P was not a multiple of 8, but that problem is no longer manifest with the current system software. Compared to the other platforms, Perlmutter and Polaris (GPUDirect in green dashed lines), demonstrate the best *per rank* performance provided that they are using GPUDirect with Slingshot 10. (Slingshot 11, newly installed on Perlmutter, yields a 1.4 1.5× improvement in MPI bandwidth but for reasons yet unresolved yields a slow-down for NekRS at certain node counts.)

It is important to point out that these strong-scaling plots start from a high level of performance. NekRS currently leverages extensive tuning of several key FP64 and FP32 kernels in libParanumal, including the standard spectral element Laplacian matrix-vector product, local tensor-product solves using fast diagonalization, and dealiased evaluation of the advection operator on a finer set of quadrature points. These kernels, which are described in a forthcoming article [37], are sustaining up to 3 TFLOPS FP64 and 5 8 TFLOPS FP32, per GPU or GCD. At the strong-scale limit, with MPI overhead, NekRS is sustaining ≈ 1 TFLOPS per rank (i.e., per GPU or GCD).

An important figure of merit is $n_{0.8}$, which is the value of $n\ P$ at which the simulation realizes 80% parallel efficiency. From the second row, right, we see that $n_{0.8} = 2\ 5$M for Perlmutter/Crusher and 2M for Polaris (GPUDirect in green dashed lines) For Polaris without GPUDirect (magenta solid line) we find $n_{0.8} = 4\ 5$M. The plot on row 3, left, indicates that a remarkably small t_{step} value of 0.015 seconds per step is realizable on Polaris, albeit at 25% efficiency.

The plot on the last row, left, shows that the time in the compute-intensive, communication-minimal advection update strong-scales quite well, as would be expected. Aside from the anomalous behavior of (earlier-OS-based) Crusher for P not a multiple of 8, the curves for the more modern architectures collapse to nearly the same performance. This kernel is sustaining 3 4 TFLOPS FP64 on these architectures (without communication). In contrast, the last row, right, shows the performance for the communication-intensive coarse grid solve, which is performed using Hypre on the host CPUs. Here, both Crusher and Polaris show relatively poor performance at small values of $n\ P$. We remark also that using Summit's *large memory nodes* generated a distinct (and worse) performance curve different than the standard 16GB nodes.

Figure 2 shows similar scaling metrics as in Figure 1 for 170-layer case across the platforms of Crusher, Summit, Perlmutter and Polaris, except that we show two different rocm versions using `rocm/5.1.0` and `rocm/5.2.0` with `cray-mpich/8.1.19`. Here we observe `rocm/5.1.0` is slightly faster performance compared to `rocm/5.2.0` with significant speedup in coarse grid solve. No GPUDirect option on Polaris (magenta solid lines) shows severe performance degradation as the number of ranks increases. We also observe that $n_{0.8} =$ 5M for Perlmutter, Crusher and Polaris (GPUDirect in green dashed lines) as the number of ranks

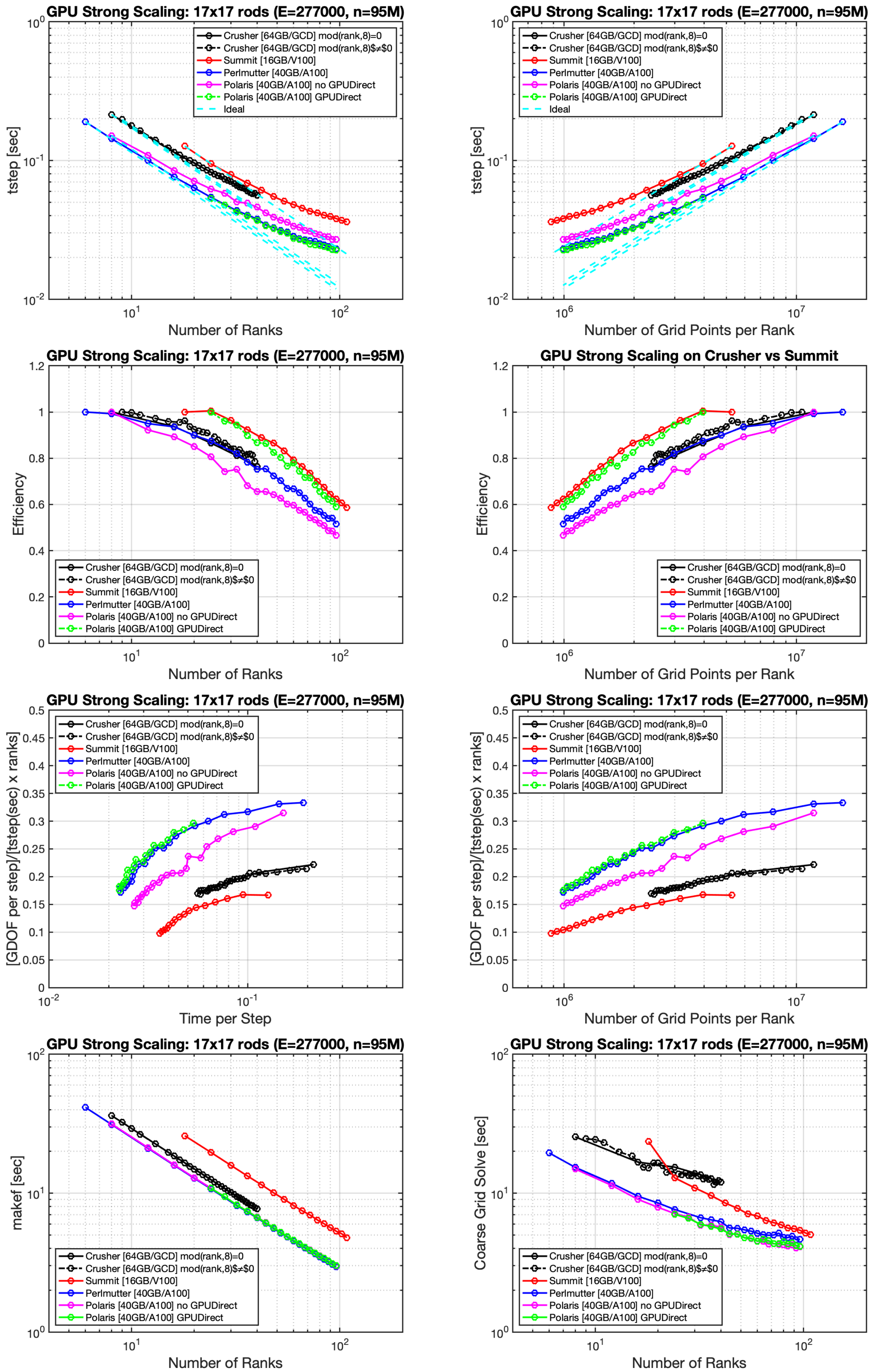


Figure 1: Strong-scaling on Crusher, Summit, Perlmutter and Polaris (17×17 rod bundle with 10 layers).

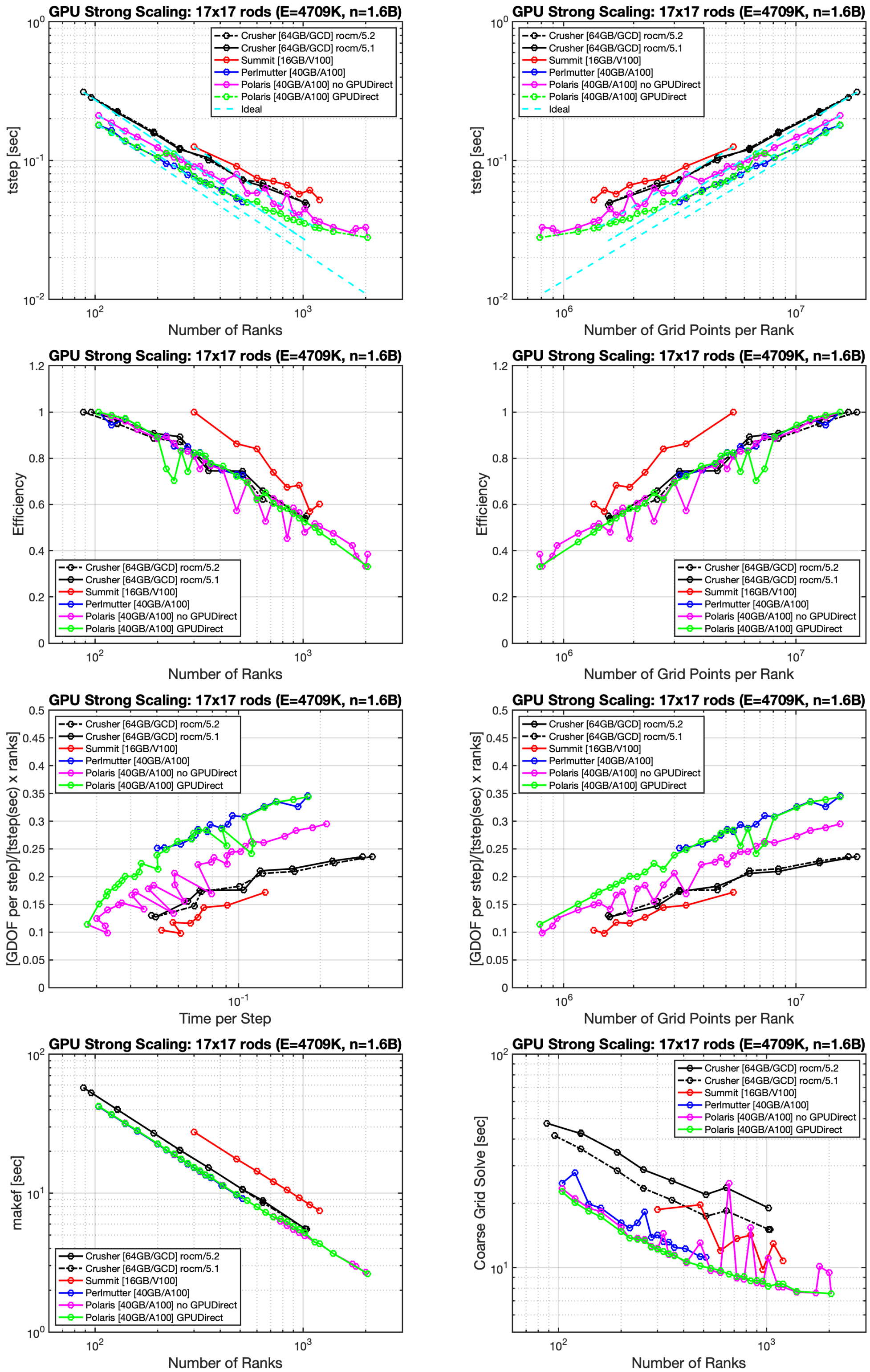


Figure 2: Strong-scaling on Crusher, Summit, Perlmutter and Polaris (17×17 rod bundle with 170 layers).

increases.

Frontier vs. Crusher. In collaboration with OLCF, the CEED team performed scaling studies on Frontier using NekRS version 22.0. Simulations on Frontier were run by John Holmen at OLCF while those on Crusher were run by the CEED team. On Frontier, `rocm/5.1.0` and `cray-mpich/8.1.17` were used. On Crusher, simulations were performed with variation of versions such as `rocm/5.1.0`, `rocm/5.2.0`, `cray-mpich/8.1.16` and `cray-mpich/8.1.19`. On Crusher, `rocm/5.1.0` is faster than `rocm/5.2.0`. We observe that the performance on Frontier is better than that on Crusher.

We consider ExaSMR's 17×17 rod-bundle geometry and extend the domain in streamwise direction with 10, 17, 170 and 6340 layers keeping the mesh density same. In this report, we present the 170-layer case, which corresponds to 4.7 million spectral elements of order $N = 7$, for a total of $n = 4\ 7\text{M} \times 7^3 = 1\ 6\text{B}$ grid points.

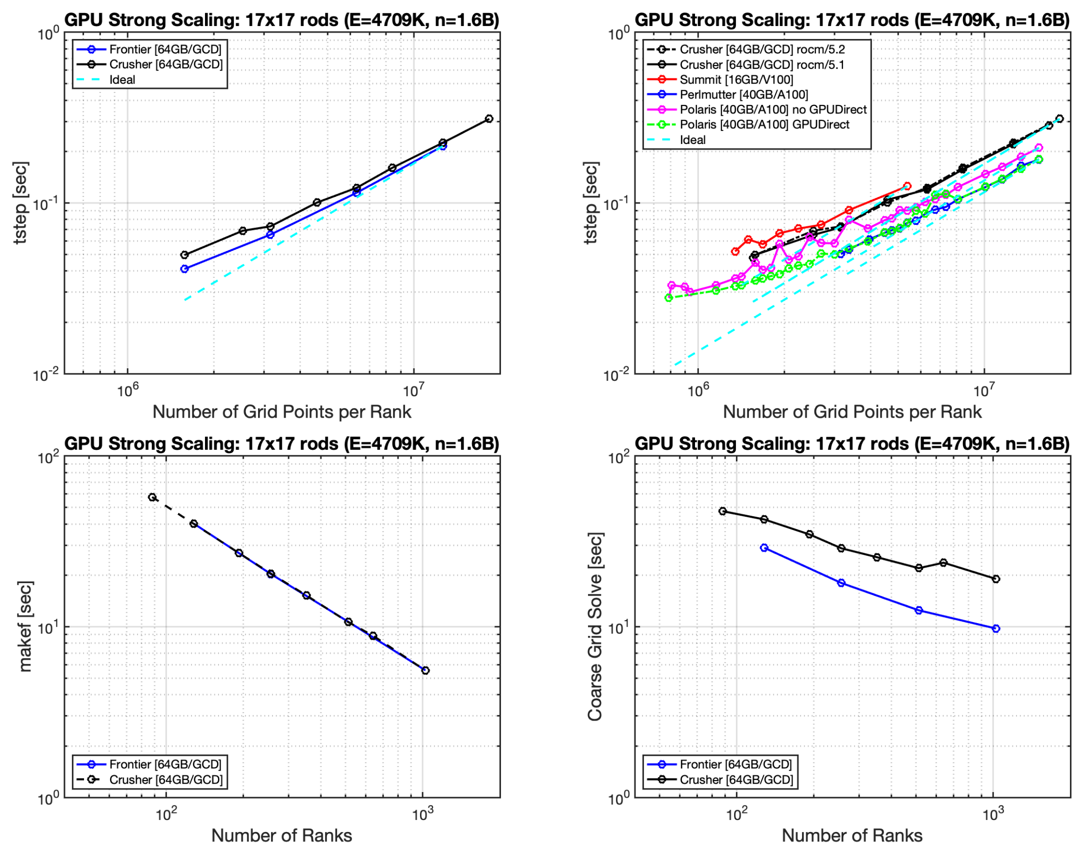


Figure 3: Strong-scaling on Frontier (cray-mpich/8.1.17, rocm/5.1.0) and Crusher (cray-mpich/8.1.19, rocm/5.2.0) for rod17 × 17 with 170 layers ($E = 27700 \times 170$). **Upper Left:** time-per-step vs. n/P for Frontier and Crusher **Upper Right:** Comparative time-per-step vs. n/P for Crusher (MI250X), Summit (V100), Perlmutter (A100), and Polaris (A100). **Lower Left:** Time in the work-intensive advection operator for Frontier and Crusher. **Lower Right:** Time in the communication-intensive coarse-grid solve for Frontier and Crusher.

Figure 3 demonstrates the scaling performance on Frontier in comparison to that on Crusher and to Summit, Perlmutter, and Polaris. The upper-left plot shows the time-per-step vs. the number of points per MPI rank, $n\ P$, where P is the number of GCDs. We run a single MPI rank per GCD. The plot indicates that Frontier is slightly faster than Crusher for this case. The lower left plot shows that Frontier and Crusher deliver the same performance on the compute-intensive `makef` kernel, which evaluates the advection term for the Navier-Stokes equations. By contrast, the lower right plot shows Crusher is a bit slower for the communication-intensive coarse-grid solve, which is part of the multigrid preconditioner for the pressure-Poisson problem. The solve is performed on the host CPUs using algebraic multigrid with Hypre. We remark that Frontier realizes 80% parallel e ciency at n $P \approx 3$ million points per rank, which would normally be viewed as the strong-scale limit for NekRS simulations on this platform.

Finally, returning to the top right gure, which is similar to the top left, we see that Crusher is faster than Summit, but not quite as fast as the A100-based Perlmutter (NERSC) and Polaris (ALCF) platforms. Moreover, the plot shows a dashed black line which are times measured several months ago on Crusher when P was not a multiple of 8. In these cases, performance was about half of that when P is a multiple of 8. This issue has now been resolved and all the Crusher results collapse to the faster (lower) line. Similarly, early results on Polaris without GPUDirect support (the magenta line) have been superseded by improved results (green line) now that GPUDirect is available on Polaris. In this case, Polaris and Perlmutter agree, *when using Slingshot 10* for the network. The recent upgrade to Slingshot 11 on Perlmutter has led to regressed performance (as much as 4×) for NekRS at certain values of P. That issue is under investigation.

2.2 AMR-Wind and NekRS performance on Summit and Crusher for ExaWind

In collaboration with the ExaWind team (Michael Sprague, Michael Brazell and Matthew Church eld at NREL), we examined large-eddy-simulation modeling approaches [35], and computational performance of two open-source computational uid dynamics codes for the simulation of atmospheric boundary layer (ABL) ows that are of direct relevance to wind energy production. The rst is NekRS, a high-order, unstructured-grid, spectral element code. The second, AMR-Wind, is a block-structured, second-order nite-volume code with adaptive-mesh-re nement capabilities. The objective of this study is to co-develop these codes in order to improve model delity and performance for each. These features will be critical for running ABL-based applications such as wind farm analysis on advanced computing architectures. To this end, we investigated the performance of NekRS and AMR-Wind on the OLCF Summit, using 4 to 800 nodes (24 to 4,800 NVIDIA V100 GPUs), and Crusher, using (18 to 384 GCDs on AMD MI250X GPUs). We compared strong- and weak-scaling capabilities, linear solver performance, and time to solution. We also identi ed leading inhibitors to parallel scaling. Further detailed studies are reported in [28].

Strong Scaling Test Sets			
Domain size	Grid Points (n)	x (m)	t (s)
[400 m]^3	512 × 512 × 512	0.78	.062500
[400 m]^3	1024 × 1024× 1024	0.39	.031250
[400 m]^3	2048 × 2048× 2048	0.19	.015625

Weak Scaling Test Sets			
Domain size	Grid Points (n)	x (m)	t (s)
400 m × 400 m × 400 m	512 × 512 ×512	0.78	.0625
800 m × 800 m × 400 m	1024 × 1024 ×512	0.78	.0625
1600 m × 1600 m × 400 m	2048 × 2048 ×512	0.78	.0625
3200 m × 3200 m × 400 m	4096 × 4096 ×512	0.78	.0625

Table 1: Problem setup for strong and weak scaling studies.

Here we compare performance and tuning for the two codes. For each case, the codes use the same spatial resolution, x, and timestep size, t. Each code uses iteration tolerances of 10^{-4} and 10^{-6} for the respective 2-norm residuals of the pressure-Poisson and velocity-Helmholtz problems. For purposes of timings, we use the solution at 6 hours as an initial condition in each case in order to ensure that performance studies are done over a timeframe in which the solutions have a representative turbulent ow. Table 1 provides a summary of the test parameters, in physical units, that are used for the strong- and weak-scaling studies. The spectral element cases use $8th$-order polynomial basis ($N = 8$) with a number of gridpoints given by $n = EN^3$. For these cases we take x to be the average grid spacing in each direction (i.e., 400 m / $n^{\frac{1}{3}}$). For the weak-scale study, the domain height is xed at 400 m while the dimensions are increased in the x and y directions as n is increased. In order to avoid initial transient behavior, the average (wall) time per step, t_{step}, in seconds is measured over steps 101 200.

We begin with performance optimization, pro ling analysis, and CPU versus GPU comparisons. NekRS GPU performance tuning on Summit is demonstrated in detail in [16, 29]. The base libParanumal kernels

AMR-Wind: Performance progress with AMReX library updates Last 200 steps averaged from 1000-step run on 8 nodes using $n = 512^3$						
	Old Version		Intermediate Version		New Version	
	(s)	[%]	(s)	[%]	(s)	[%]
Wall time per timestep	3.3200e-01	100	2.4100e-01	100	2.2500e-01	100
Advection	2.9814e-02	8.98	2.6920e-02	11.17	2.8687e-02	12.75
MAC Projection	6.2582e-02	18.85	6.2660e-02	26.00	6.3135e-02	28.06
Pressure Solve	7.3671e-02	22.19	7.3481e-02	30.49	6.3180e-02	28.08
Velocity Solve	1.1401e-01	34.34	3.9669e-02	16.46	3.9307e-02	17.47
Scalar Solve	3.4827e-02	10.49	2.2148e-02	9.19	1.5930e-02	7.08
Fillpatch	1.5538e-02	4.68	1.4990e-02	6.22	1.5030e-02	6.68

Table 2: AMR-Wind performance optimization.

have their origins in the work of Warburton and co-workers [8, 33, 9, 2]. A key algorithmic component is the Chebyshev-accelerated Schwarz-based p-multigrid for the pressure solve [31], which is performed in 32-bit precision (e.g., as in [15]) to reduce injection-bandwidth pressure on the Summit network interface cards. Communication for the nearest-neighbor communication (direct-sti ness summation in the nite element or spectral element context [10]) is overlapped with computation whenever it proves to be e ective, which can yield as much as 10 15% savings in NS applications. At the strong-scale limit of $\approx$ 2M points per GPU, there are enough points interior to each rank's subdomain to balance out the communication overhead for the gather-scatter exchanges, at least at the ne-mesh level evaluations. For the coarser p-multigrid levels, it is not always the case that one can cover the communication with work. When initializing the communication kernels for each level of the p-multigrid solver, NekRS selects the fastest of several available communication strategies (e.g., overlapping, pack-on device or host, GPU direct or via the host), which are determined by timing tests during runtime setup. These unit tests also report the observed messaging bandwidth and thus provide insight into possible anomalous system behavior, which is useful when porting relatively new code to relatively new and unknown HPC platforms. For example, from existing log les, we were able to compare the observed bandwidth for several gather-scatter exchanges on NERSC's Perlmutter platform before and after a network update from Cray's Slingshot 10 to Slingshot 11, as shown below.

```
SS10:
pw+device MPI: 7.37e-05s / bi-bw:  54.5GB/s/rank
pw+device MPI: 5.16e-05s / bi-bw: 100.2GB/s/rank
pw+device MPI: 3.84e-05s / bi-bw:  33.6GB/s/rank
pw+host   MPI: 2.46e-05s / bi-bw:   3.6GB/s/rank

SS11:
pw+device MPI: 4.38e-05s / bi-bw:  91.8GB/s/rank
pw+device MPI: 3.47e-05s / bi-bw: 148.8GB/s/rank
pw+device MPI: 2.74e-05s / bi-bw:  47.2GB/s/rank
pw+host   MPI: 1.66e-05s / bi-bw:   5.4GB/s/rank
```

Here, SS10 indicates Slingshot 10, and SS11 indicates Slingshot 11, which shows about a 1.5$\times$ improvement over SS10. The listings also show which communication mode was used. We see that `pw+device`, which stands for pairwise device-to-device exchange (i.e., via GPU-direct) is used in most instances. The `pw+host`, which indicates the use of pairwise exchanges via the host, is used only in the case of many short messages, which is typically the scenario at the coarsest levels of the p-multigrid solver.

Over the course of the collaboration, AMR-Wind realized a 1.4$\times$ speedup with some improvements derived through AMReX library updates. The performance progress is demonstrated in Table 2, where the rows present a timing breakdown of a typical ow time step. Advection involves predicting and forming the advection term using Godunov PPM WENO. MAC projection is a Poisson equation linear solve with a 7-point stencil that ensures that the face velocities are divergence free. The pressure solve is a Poisson equation linear solve with a 27-point stencil that approximately corrects the cell velocity to be divergence free at the end of the time step. Velocity and scalar solve are Helmholtz equations with a 7-point stencil, and Fillpatch performs all communication within and across processors outside of the linear solver communication. In the table, the *old version* is AMR-Wind using AMReX from April 2021. The *intermediate version* is the same source code but with improvements to the linear solver settings. In particular, the components of the

Nsight-Compute Profiling: CUDA Kernel Statistics 11 nodes (66 GPUs), $n/P = 2.03M$, $n = 512^3$, 2000 steps					
NekRS					
Time (%)	Total Time (ms)	Instances	Average (μs)	Name	Remark
7.2	1438.604	3327	432.402	`subCycleStrongCubatureVolumeHex3D`	dealiased vel. adv.
6.6	1320.502	8002	165.021	`gatherScatterMany_doubleAdd`	FP64 local gather-scatter
5.1	1017.806	8144	124.976	`packBuf_doubleAdd`	FP64 packing for gs
5.0	1009.247	3533	285.663	`subCycleStrongCubatureVolumeHex3D`	dealias scalar adv.
4.7	935.298	251	3726.289	`scatterMany_double`	FP64 scatter
4.4	879.495	8144	107.993	`unpackBuf_doubleAdd`	FP64 gather
3.3	667.776	3192	209.203	`subCycleRKUpdate`	RK4 vector update
2.9	577.150	850	679.000	`ellipticStressPartialAxCoeffHex3D`	viscous op. eval.
2.4	480.331	2788	172.285	`ellipticPartialAxHex3D`	pressure op. eval.
AMR-Wind					
Time (%)	Total Time (ms)	Instances	Average (μs)	Name	Remark
15.0	1890.142	142823	13.234	`fab_to_fab`	array box local copy
9.9	1256.128	12800	98.148	`MLNodeLaplacian::Fsmooth`	multigrid smoother
6.9	873.629	24800	35.227	`amrex::Copy`	multiple array box parallel copy
5.9	738.200	5600	131.821	`MLABecLaplacian::Fapply`	Laplacian op. eval.
4.5	564.742	43200	13.072	`MLPoisson::Fsmooth`	multigrid smoother
3.5	438.271	6800	64.451	`MultiFab::LinComb`	vector-vector addition
3.1	394.024	3200	123.132	`MLABecLaplacian::normalize`	normalize solution
3.0	384.391	800	480.488	`godunov::compute_fluxes`	advection momentum
2.9	359.910	800	449.887	`godunov::compute_fluxes`	advection scalar
2.7	344.575	11800	29.201	`MultiFab::Xpay`	vector-vector addition
2.4	303.2	29850	11.085	`FabArray::setVal`	set value of array box

Table 3: CUDA kernel statistics from NVIDIA® Nsight™ profiler using `nsys profile --stats=true -t nvtx,cuda`.

momentum equations are solved separately instead of as a coupled tensor solve. The velocity and scalar (temperature) linear systems are solved by using bi-conjugate gradient iteration instead of a full geometric multigrid approach. In Table 2 we see that these optimizations reduce the velocity solve time by almost 3× (.114 s to .039 s) and the scalar solve time by 1.5× (.035 s to .022 s). The *new version* is AMR-Wind based on AMReX from 2022 with the same improved linear system settings. Here the scalar solve improves by another factor of 1.5. and the pressure solve is reduced from .0073 s to .0063 s per step.

For AMR-Wind, Table 3 more clearly indicates the elliptic solves as leading cost contributors. This cost is also reflected in Figure 4, where the two largest contributors to run time are the pressure solve and the MAC projection onto a divergence-free space. In fact, these plots show that the requirement of two Poisson-like solves for AMR-Wind is the principal cause for discrepancy in run-time between the two codes. MAC projection is solved using geometric multigrid. While not necessary, it does provide more robustness and increases the stability of the scheme to CFL=2. If it did not require the MAC step, AMR-Wind would be faster on 4 nodes than NekRS. `pressureSolve` is a Poisson solve that is used at the end of the timestep to form an approximate divergence-free velocity at the cell center; it is a node-based 27 point stencil, and the linear system is solved using geometric multigrid. `scalarSolve` and `velocitySolve` are both Helmholtz solves that are cell-based 7 point stencils, the `scalarSolve` advances in time the potential Temperature equation and `velocitySolve` is three separate solves to advance each of the momentum equations in time. BiCG is used to solve all of the linear Helmholtz subproblems. The advection terms in the governing equations are discretized using a Godunov WENO-Z scheme to provide these terms on the cell faces at time $t^{n+\frac{1}{2}}$. Other function calls comprise source term calculations, boundary conditions, planar averaging, communication (excluding linear solve communication), linear solve setup, and copying solution arrays.

For AMR-Wind on both Summit and Crusher, the time per step, t_{step}, decreases with increasing node count as each component of the timestep takes less time. Both Poisson solves, however, take a higher percentage of the time step as P is increased, which reflects the communication-intensive nature of the Poisson problem. Overall, Crusher is providing better performance than Summit. This is partly because there are more GPUs per node (8 versus 6) but also because the mesh decomposition has better load balancing for

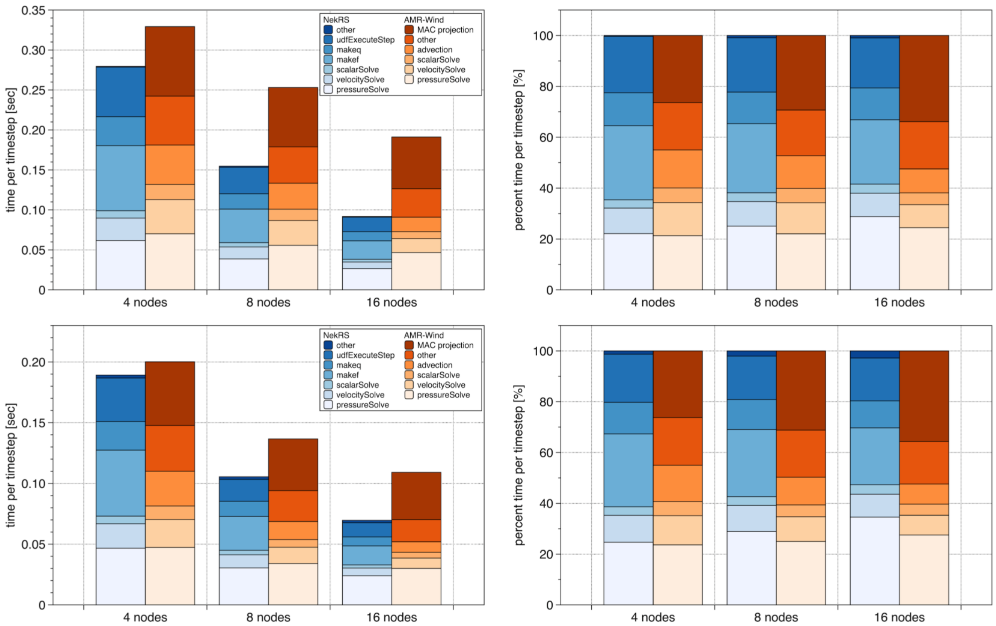


Figure 4: NekRS vs AMR-Wind GPU cost breakdown on Summit (top) and Crusher (bottom), using $n = 512^3$ and 2000 steps.

AMR-Wind. A problem size of 512^3 is more easily partitioned by 8 GPUs/node versus 6 GPUs/node. With 16 Crusher nodes (128 GPUs) a time per timestep of t_{step}=0.11 s is achieved with AMR-Wind. Further scaling out with Summit the lowest time per timestep was 0.128 s on 128 Summit nodes (768 GPUs), as discussed below.

For NekRS, we start the GPU analysis with NVIDIA's pro ling tools. Table 3 summarizes the kernel-level metrics for the critical kernels, which are identi ed with NVIDIA's Nsight Systems. At this granularity, the table indicates that the bulk of the time for NekRS is spent evaluating the dealiased advection operator (`subCycleStrongCubatureVolumeHex3D`) both for the velocity vectors and for the temperature. Other leading consumers are the gather-scatter operations. Largely missing from this table for NekRS is the time spent in the pressure preconditioner, which is separated across many kernels for the various levels of p-multigrid. Each NekRS job tracks basic runtime statistics using a combination of MPI `Wtime` and `cudaDeviceSynchronize` or CUDA events. These are output every 500 time steps unless the user speci es otherwise. From these, we collect aggregate timing breakdowns, roughly following the physical substeps of advection, pressure, and viscous- thermal-updates, plus tracking of known communication bottlenecks such as the pMG coarse-grid solve for the pressure preconditioner. Figure 4 shows the cost breakdown for this type of analysis over node counts ranging from 4 to 16. At lower node counts, the bulk of the NekRS time is spent in the `makef` and `makeq` (advection) routines, which are respectively responsible for setting up the right-hand-sides for the momentum and energy equations. To allow a larger CFL, the ABL simulations use characteristics-based timestepping, which involves solving a sequence of hyperbolic subproblems on the interval $[t^{n-2}\ t^n]$ (one for each velocity component and one for temperature) [27, 30]. Each subproblem takes several substeps using the dealiased advection operator, which performs quadrature on a $11 \times 11 \times 11$ grid in each element. These substeps are thus compute-intensive but not communication intensive, so they scale relatively well. `velocitySolve` and `scalarSolve`, which involve communication-free diagonal preconditioning for conjugate gradient approach, show similar scaling behavior. As with AMR-Wind, we see clearly in Figure 4 that the pressure solve not scale as well as the other components.

We remark that Figure 4 indicates a signi cant amount of time is spent in `udfExecuteStep`. The majority of that cost results from the recently adopted mean- eld eddy viscosity model, which requires several planar averages per time step and is currently implemented as a user-de ned function. For these calculations, which have low pressure and velocity iteration counts, the frequently called planar average utility has a significant impact on runtime (about 20%). Planar averaging is typically a post-processing operation that is not performed on every step, but clearly it will need to be optimized in this LES application.

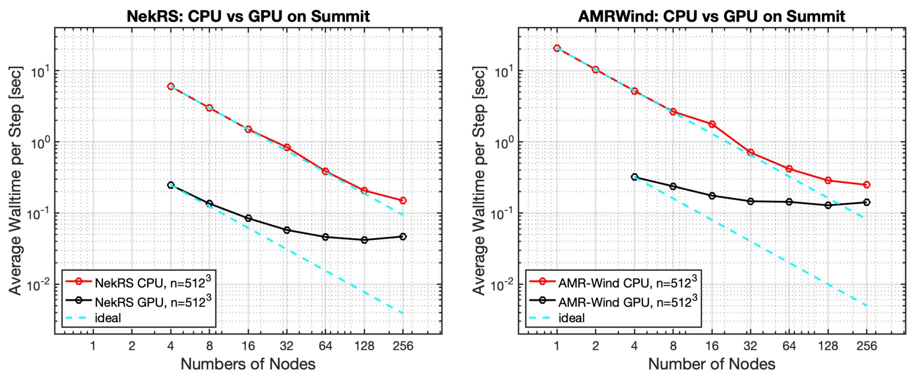


Figure 5: NekRS and AMR-Wind: CPU vs. GPU performance on Summit: 100 steps average from 200 step runs for $n = 512^3$.

Figure 5 shows CPU and GPU strong-scaling performance for each code on Summit. The upper figures show standard time vs. node-count plots, which clearly indicate that it is easier to strong-scale on the CPU. On Summit, however, that point is moot given that one needs 128 nodes using a CPU-only configuration in order to get to the same time-per-step as using 4 nodes with 6 GPUs each (i.e., *a factor of 32 difference in required node-hours to do the same work*).

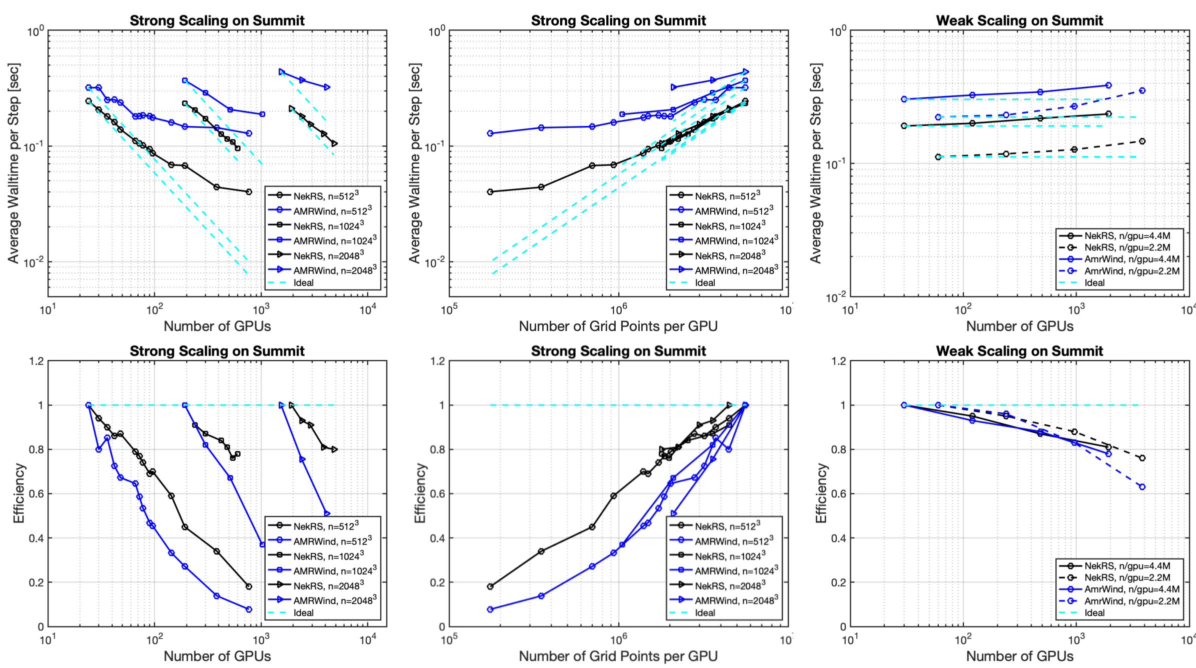


Figure 6: NekRS vs. AMR-Wind strong and weak scaling on Summit GPUs.

We next consider GPU-only performance on Summit using a single V100 per MPI rank. Figure 6, top, shows performance in terms of t_{step} for strong scaling as a function of the number of GPUs, P, in the left column and as a function of number of points per rank, n P, in the center column. Weak-scaling performance is presented in the right column. The wall-time figure also shows the ideal speed-up curves scaling as P^{-1}.

Strong-Scaling on Summit GPU, $n = 512^3$, $x = 0.78$ m, $t = 6.25$e-2s, $= [400$ m $\times$ 400 m $\times$ 400 m$]$															
			NekRS						AMR-Wind						
node	gpu	n/gpu	v_i	p_i	T_i	t_{step}	P_{eff}	r_t	v_i	p_i	T_i	t_{step}	P_{eff}	r_t
4	24	5.5924e+06	2	1.81	1	2.44e-01	100	3.90	2	2	2	3.19e-01	100	5.10
8	48	2.7962e+06	2	1.82	1	1.39e-01	87	2.22	2	2	2	2.37e-01	67	3.80
11	66	2.0336e+06	2	1.85	1	1.11e-01	79	1.78	2	2	2	1.79e-01	64	2.87
16	96	1.3981e+06	2	1.90	1	8.66e-02	70	1.38	2	2	2	1.75e-01	45	2.80
24	144	9.3207e+05	2	2.00	1	6.87e-02	59	1.09	2	2	2	1.60e-01	33	2.56
32	192	6.9905e+05	2	2.00	1	6.77e-02	45	1.08	2	2	2	1.46e-01	27	2.34
64	384	3.4953e+05	2	2.00	1	4.40e-02	34	0.70	2	2	2	1.43e-01	13	2.30
128	768	1.7476e+05	2	2.00	1	4.02e-02	18	0.64	2	2	2	1.28e-01	7.7	2.05
256	1536	8.7381e+04	2	2.00	1	3.60e-02	10	0.57	2	2	2	1.41e-01	3.5	2.26

Strong-Scaling on Summit GPU, $n = 1024^3$, $x = 0.39$ m, $t = 3.125$e-2s, $= [400$ m $\times$ 400 m $\times$ 400 m$]$														
			NekRS						AMR-Wind					
node	gpu	n/gpu	v_i	p_i	T_i	t_{step}	P_{eff}	r_t	v_i	p_i	T_i	t_{step}	P_{eff}	r_t
32	192	5.5924e+06	1	1.4	1	2.34e-01	100	7.50	2	2	2	0.369	100	11.82
40	240	4.4739e+06	1	1.3	1	2.04e-01	91	6.54	2	2	2	0.402	73	12.86
50	300	3.5791e+06	1	1.4	1	1.72e-01	87	5.50	2	2	2	0.288	82	9.21
70	420	2.5565e+06	1	1.4	1	1.27e-01	84	4.06	2	2	2	0.303	56	9.72
80	480	2.2370e+06	1	1.3	1	1.15e-01	81	3.68	2	2	2	0.301	49	9.65
86	512	2.0972e+06		-		-	-		2	2	2	0.206	67	6.60
90	540	1.9884e+06	1	1.3	1	1.08e-01	76	3.47	2	2	2	0.217	60	6.95
100	600	1.7896e+06	1	1.3	1	9.57e-02	78	3.06	2	2	2	0.219	54	7.01

Strong-Scaling on Summit GPU, $n = 2048^3$, $x = 0.39$ m, $t = 1.5625$e-2s, $= [400$ m $\times$ 400 m $\times$ 400 m$]$														
			NekRS						AMR-Wind					
node	gpu	n/gpu	v_i	p_i	T_i	t_{step}	P_{eff}	r_t	v_i	p_i	T_i	t_{step}	P_{eff}	r_t
256	1536	5.5924e+06		-		-	-		2	2	2	0.437	100	34.99
320	1920	4.4739e+06	1	1.16	1	2.10e-01	100	16.8	2	2	2	0.485	72	38.80
400	2400	3.5791e+06	1	1.20	1	1.80e-01	93	14.4	2	2	2	0.370	76	29.62
480	2880	2.9826e+06	1	1.22	1	1.54e-01	91	12.3	2	2	2	0.402	58	32.16
640	3840	2.2370e+06	1	1.25	1	1.28e-01	81	10.3	2	2	2	0.440	40	35.26
683	4096	2.0972e+06		-		-	-		2	2	2	0.321	51	25.69
800	4800	1.7896e+06	1	1.18	1	1.05e-01	80	8.4	2	2	2	0.390	36	31.23

Table 4: NekRS GPU vs. AMR-Wind GPU strong-scaling performance study.

The lower plots show parallel e ciency,

$$_P := \frac{t_0 P_0}{t_{step} P} \tag{1}$$

where P_0 is the smallest value of P that will hold the given problem and t_0 is the t_{step} value corresponding to P_0.

We see that at the lower resolution of $n = 512^3$, the performance of the two codes is within a factor of 2 of each other out to $P = 78$. From the e ciency gures we can see that both curves have dropped below 80% e ciency by that point, so a more realistic point of comparison would be at $P = 66$ given that users would typically not run this relatively small case on $P > 66$. We note that $P = 66$ corresponds to $n \; P = 2$M, which is a typical strong-scaling limit for NekRS on current-generation GPU platforms.

The center column in Figure 6 replots the strong-scaling information with $n \; P$ as the independent variable. Here we see a collapse of each code's strong-scale data into a single curve, particularly for NekRS. The e ciency plot, lower-center, clearly shows the $n \; P = 2$M mark as the 80% parallel e ciency point for NekRS. The AMR-Wind wall-time curves, upper center, are not as tightly grouped, particularly for the large problem sizes on large processor counts. It is tempting to speculate that this increased cost is due to an increase in iteration count, but Tables 4 and 5 show that is not the case, since each solver requires only two iterations per timestep for each of the problems. An important feature of AMR-Wind is that it generally performs better if P is a power of 2. At the critical point of $n \; P = 2$M, NekRS is only a factor of 1.6 faster than AMR-Wind for the $n = 512^3$ case.

Figure 6, right, shows weak-scaling results for $n \; P = 2$ 2M and 4.4M. For the heavily loaded cases, AMR-Wind is within a factor of 1.6 of NekRS, but this gure increases to roughly a factor of 2 for the 2.2M

Weak-Scaling on Summit GPU, $x = 0.78$ m, $t = 6.25\text{e-}2$s															
				NekRS						AMR-Wind					
node	gpu	n	n/gpu	v_i	p_i	T_i	t_{step}	P_{eff}	r_t	v_i	p_i	T_i	t_{step}	P_{eff}	r_t
5	30	$512^2\times$ 512	4.4e+06	1	1.5	1	0.191	100	3.8	2	2	2	0.303	100	4.8
20	120	$1024^2\times$ 512	4.4e+06	1	1.7	1	0.200	95	4.0	2	2	2	0.326	93	5.2
80	480	$2048^2\times$ 512	4.4e+06	1	1.9	1	0.218	87	3.8	2	2	2	0.344	88	5.5
320	1920	$4096^2\times$ 512	4.4e+06	1	2.4	1	0.235	81	4.7	2	2	2	0.386	78	6.2
				NekRS						AMR-Wind					
node	gpu	n	n/gpu	v_i	p_i	T_i	t_{step}	P_{eff}	r_t	v_i	p_i	T_i	t_{step}	P_{eff}	r_t
10	60	$512^2\times$ 512	2.2e+06	1	1.5	1	0.112	100	2.2	2	2	2	0.223	100	3.6
40	240	$1024^2\times$ 512	2.2e+06	1	1.7	1	0.118	95	2.4	2	2	2	0.231	96	3.7
160	960	$2048^2\times$ 512	2.2e+06	1	2.1	1	0.127	88	2.5	2	2	2	0.269	83	4.3
640	3840	$4096^2\times$ 512	2.2e+06	1	2.5	1	0.147	76	2.9	2	2	2	0.352	63	5.6

Table 5: NekRS GPU vs. AMR-Wind GPU weak-scaling performance study with fixed mesh density and resolution per GPU.

points-per-GPU case. The weak-scale e ciency reaches 80% at around $P = 2000$ GPUs for all the cases save the AMR-Wind case with n $P = 2$ 2M, which crosses the 80% mark at $P \approx 1100$.

Tables 4 and 5 provide a detailed breakdown of several of the key metrics for the code performance, including iteration counts (v_i, p_i, T_i, for the respective velocity, pressure, and temperature iterative solvers), t_{step}, parallel e ciency (P_{eff}), and the wall-time to physical-time ratio (r_t). This last quantity is of particular interest since it must be smaller than unity for weather modeling applications. We also note that P is denoted by gpu in the tables. We see from Table 4 that, for a xed value of n P, r_t e ectively doubles with each doubling of (linear) resolution. The reason for this increase is that the number of timesteps must also double whenever the number of points in each direction is doubled (for xed domain size). Throughout the table, we see that roughly two iterations are required per timestep for each of the linear solvers, indicating that the preconditioners are robust with respect to mesh size, although NekRS does show some increase in iteration count in the weak-scale results.

We remark that AMR supports block-structured adaptive mesh re nement, which means that static grids do not leverage one of its main features. It is nonetheless highly performant on this problem. Moreover, AMR-Wind has a signi cant performance boost when the number of ranks is a power of 2, as seen in Table 4 for the $n = 1024^3$ case for $P = 512$ and in the $n = 2048^3$ case for $P = 4096$. In the former case, the parallel e ciency jumps from 49% to 67% as P changes from 480 to 512. In the latter, it jumps from 40% for $P = 3840$ to 51% for $P = 4096$. These performance gains derive from the block decompositions used in AMR-Wind, which favor block sizes (and thus, processor counts) that are powers of 2.

We close with a scaling comparison of Summit and Crusher performance for NekRS in Figure 7. The upper gures show standard strong scaling as a function of the number of ranks on the left (one GPU or GCD per rank) and as a function of n P on the right. The lower plots show the timing for the `makef` kernel (left), which evaluates the nonlinear advection term and does not require communication, and for the coarse-grid solve (right), which is communication dominated. The coarse-grid problem, which has roughly E degrees of freedom (with $E = 262144$ in this case), is solved by using algebraic multigrid (hypre) on the host CPUs. The performance for these two platforms is remarkably similar.

2.3 MFEM fused GPU solvers for the DG mass operator in Laghos

GPU solvers for the mass matrix are critical to the performance of application using explicit time integration. The discontinuous Galerkin (DG) mass matrix with its block diagonal structure allows to solve small dense linear systems local to each element instead of a global linear problem. Solving local small dense linear systems avoids expensive MPI communications while also allowing to use either iterative or direct solvers on GPU, or even computing an explicit inverse. This study implemented in MFEM and in the Laghos mini-application shows that using any of the proposed local solver result in substantial solve time speedups on GPU, from 10x to 100x, when compared to a global matrix-free conjugate gradient. The solvers considered in thr study are the following:

- **Global matrix-free conjugate gradient preconditioned with Jacobi** (FCG): similar to the CEED

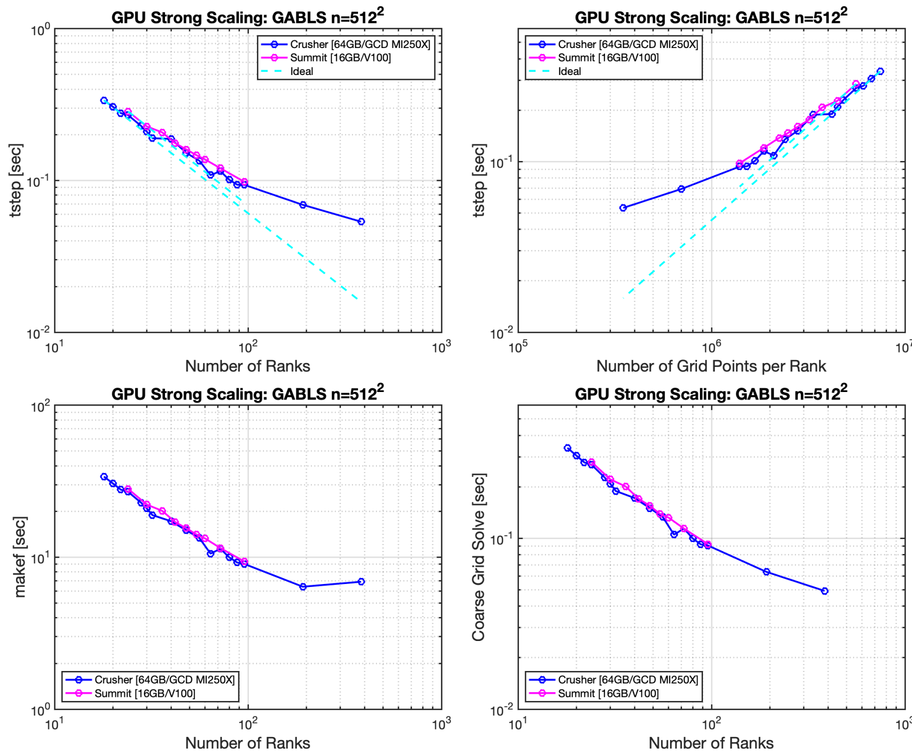


Figure 7: NekRS GPU strong-scaling comparison on Crusher and Summit.

Bake-o problem BP1 but for discontinuous Galerkin mass operator with a Jacobi preconditioner.

- **Local matrix-free *fused* conjugate gradient preconditioned with Jacobi**: (LCG-Leg with *Legendre* basis and LCG-Lob with *Lobatto* basis) contrary to the global conjugate gradient where each step of the conjugate gradient algorithm (operator evaluation, dot product, axpy) is a di erent GPU kernel; for the local conjugate gradient a single GPU kernel *fuses* all the steps of the algorithm into one kernel. As a result, this approach uses a single kernel launch instead of hundreds of them in a global conjugate gradient. Moreover, local problems are solved at the thread block level, allowing most of the computation to happen in registers and shared memory, and do not require to use device memory bandwidth.

- **Direct solvers**:
  - **Local dense LU factorization** (LU): the dense diagonal blocks are assembled on GPU with MFEM, and then a dense LU factorization implemented in MFEM is executed on device.
  - **Batched Cholesky factorization using *cusparse*** (Chol): the dense diagonal blocks are assembled on GPU with MFEM, and then a batched Cholesky factorization from NVIDIA's *cusparse* library is called.
  - **Explicit local inverse and batched product from *cublas*** (Inv): the dense diagonal blocks are assembled on the GPU with MFEM, and then explicit inverse are stored in place, and nally batched dense product from *cublas* is called.

The fused GPU solver for the discontinuous Galerkin mass matrix has been integrated into the CEED Laghos mini-application. A 3D triple point benchmark has been used to demonstrate the scaling on Lassen and Crusher, with 524,288 elements, 505 million of total degrees of freedoms, 33 million thermodynamic (L^2) degrees of freedom, using kinematic order 4 and thermodynamic order 3. Table 6 shows the two orders of magnitude speedup at order 4 that can be achieved.

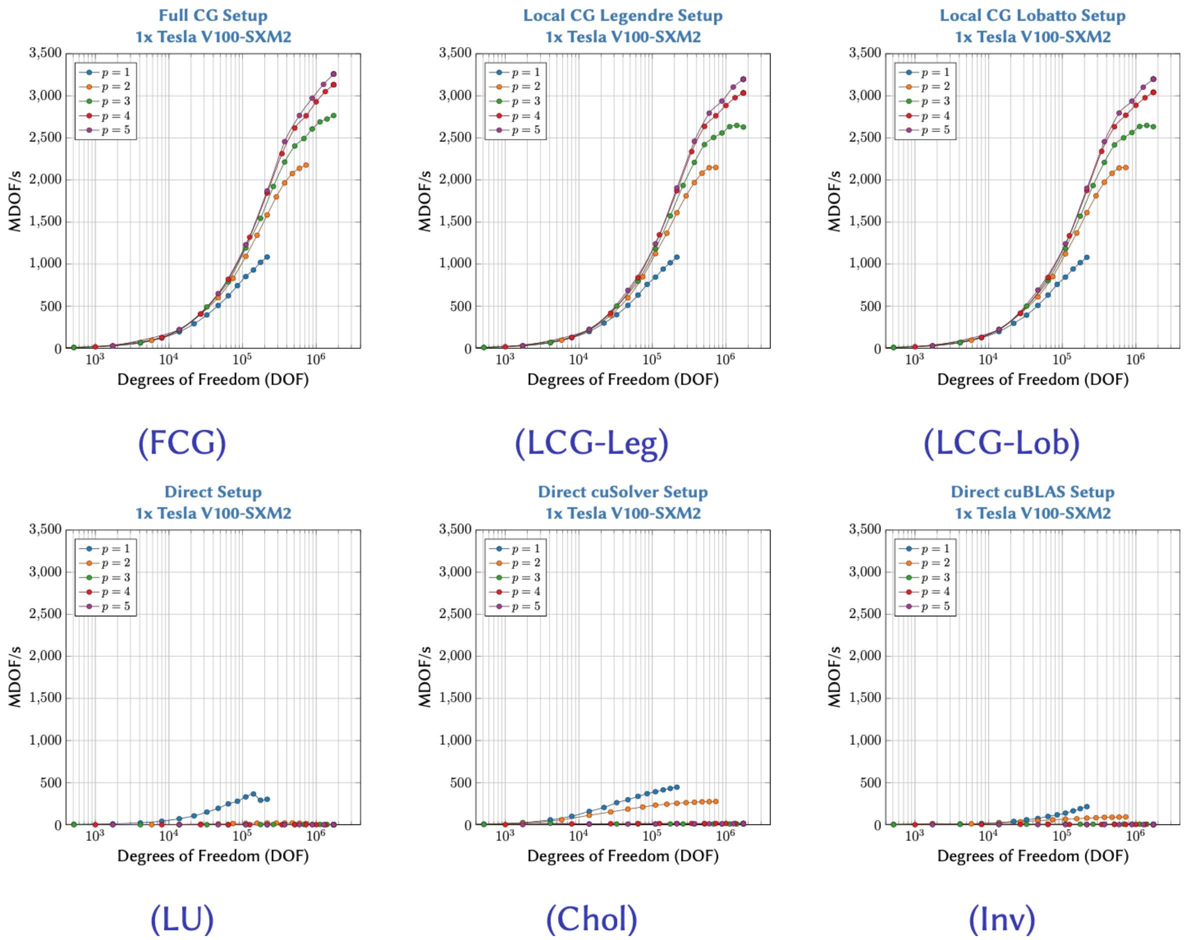


Figure 8: Comparison of the setup throughput for the different solvers. The setup throughput is the same for all conjugate gradient based solvers, either local or global; on the other hand direct solvers have a much higher cost. Direct solvers should therefore be avoided in favor of conjugate gradient solvers when the mass matrix changes over time.

	Number of GPUs	L^2 DOFs (Millions)	Not-fused Runtime(s)	Fused Runtime(s)	**Speedup**
Lassen	32	33	6.350	0.055	**120**
	512	33	3.323	0.018	**180**
Crusher	32	33	9.173	0.066	**140**
	512	33	1.876	0.008	**230**

Table 6: Two orders of magnitude Laghos speedup reached at order 4 with the fused GPU solver for the discontinuous Galerkin mass matrix.

2.4 CEED benchmarks with matrix instructions on AMD MI250X

During recent benchmarking we observed that the performance of our most heavily tuned CEED BK kernels on the AMD MI250X saturates at 1.2TB/s, and sometimes at 1TB/s. This is considerably lower than the theoretical peak of approximately 1.6TB/s. We also observed that the achieved peak throughput is correlated with the device memory read:write ratio of the kernels. Quantifying this observation involved creating additional simpli ed streaming tests for the streamParanumal benchmarking suite which allows us to

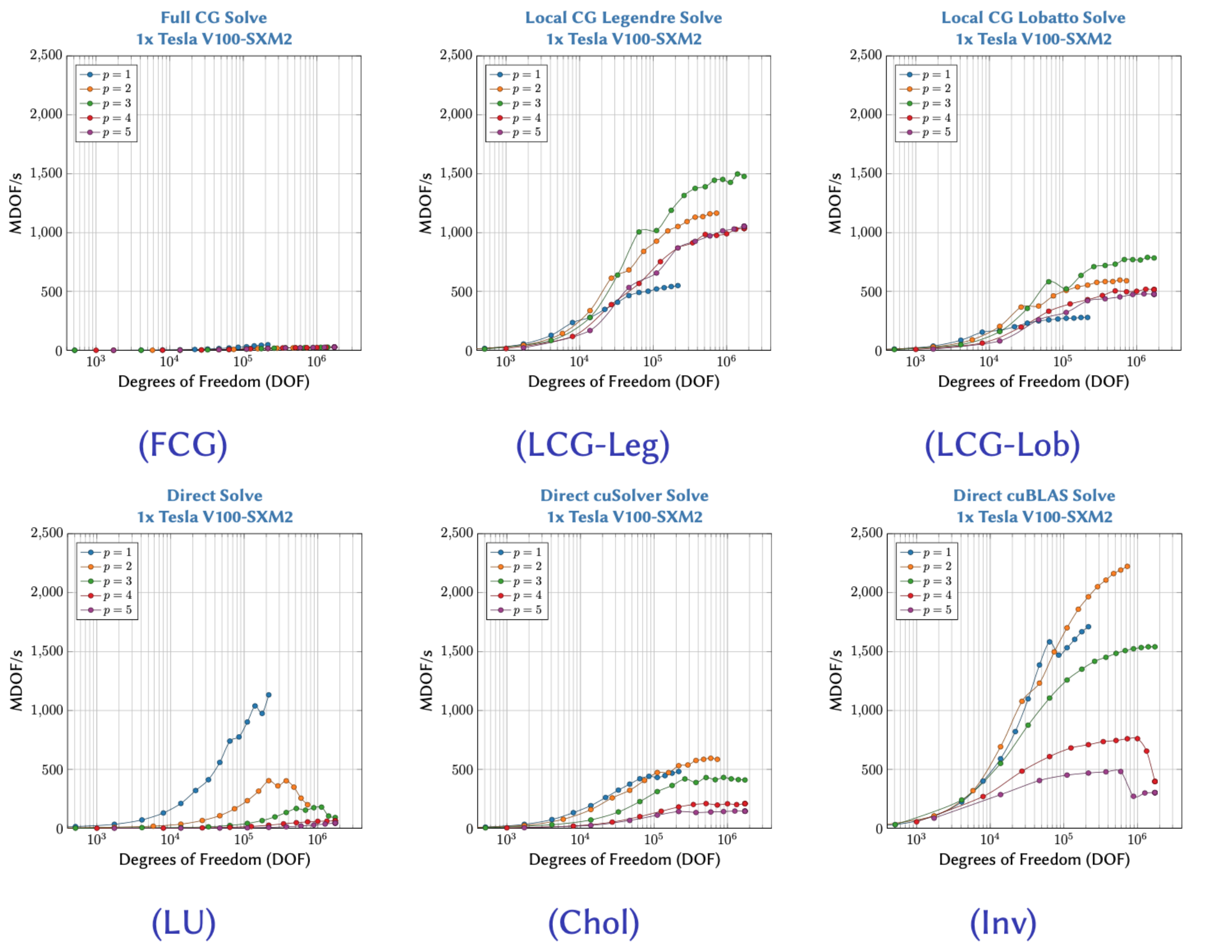


Figure 9: Comparison of the total solve throughput for the different solvers. The global conjugate gradient peak performance is around 20-30 MDOF/s, which is about two orders of magnitude slower than the best performing solver. Local conjugate gradient solvers achieve good performance for the full range of polynomial orders. Direct solvers constitute an interesting alternative, especially explicit inverse for low orders.

test di erent read:write ratios. We will subsequently refer to the asymmetric streaming benchmark as BS9 .

We created three native HIP kernel implementations for the BS9 kernel with di erent striding choices for the read and writes. These kernels were compiled at runtime using the JIT facilities of the OCCA portability library which was convenient for testing a wide range of read:write ratios. The most e cient kernel is shown in Listing 1.

The throughputs achieved for BS9 by the three kernels on a single GCD of the AMD MI250X are shown in Figure 10. There are two notable aspects of these results. First, there is considerable variation between the performance of the three di erent kernels. Second, the best performing kernel (illustrated in Listing 1) shows degradation of throughput as the read:write ratio increases. Whereas the throughput is nearly 1.4TB/s for a read:write ratio of 1:1 it degrades to about 1.26TB/s for a read:write ratio of 8:1 which closely mirrors the memory access ratio inherent to BK3 and BK5. Thus in general it is only reasonable to expect up to about 1.26TB/s for those benchmarks which is considerably lower than the theoretical memory throughput of the AMD MI250X GPU.

Performance of CEED BK3 with matrix instruction acceleration on the AMD MI250X. In this section we discuss our recent optimization campaign for the CEED BK3 kernel at high order on the AMD MI250X.

In prior CEED milestone reports we have presented results for AMD GPUs up to only polynomial degree ($N = 8$). This is in part because users oftentimes find that the sweet spot for high-order finite elements is typically around $N = 7$. However, it is also in part because there are several performance limiters intrinsic to the AMD MI200 series core architecture that can make it difficult to achieve high throughput for higher polynomial degrees.

In Figure 11 we show block diagrams for the NVIDIA GA100 streaming multiprocessor (SM) used in the NVIDIA A100 GPU and the AMD CDNA2 compute unit (CU) used in the AMD MI250X. These two architectures are quite similar apart from their L1 and shared memory cache sizes. The NVIDIA SM has more shared scratch space than the AMD CU. Thus optimizations suitable for the NVIDIA GPU that rely heavily on L1 and/or shared memory may not be as suitable for the AMD CU.

Listing 1: Listing for HIP implementation of the BS9 asymmetric streaming kernel

```
// p_NblockSize is number of threads per thread-block
// p_Nreads is number of values to read and sum
// p_Nwrites is number of times to write sum out
// dfloat is the type of the values (double for this work)
extern "C" __global__ __launch_bounds__(p_blockSize)
void bs9_v2(const dlong N,
            const dfloat* __restrict__ x,
                  dfloat* __restrict__ y){

  int n = threadIdx.x + blockIdx.x*p_blockSize*p_Nreads;

  dfloat res = 0;

#pragma unroll p_Nreads
  for(int m=0;m<p_Nreads;++m){
    dlong id = n + m*p_blockSize;
    if(id<p_Nreads*N){
      res += x[id];
    }
  }

  n = threadIdx.x + blockIdx.x*p_blockSize*p_Nwrites;

#pragma unroll p_Nwrites
  for(int m=0;m<p_Nwrites;++m){
    dlong id = n + m*p_blockSize;
    if(id<p_Nwrites*N){
      y[id] = res;
    }
  }
}
```

In [34] we described how the 12 CEED BK3 tensor contractions can be efficiently handled with a mixture of storage in registers and shared memory on NVIDIA GPUs. However, at high order this strategy is hampered by the limited available shared memory on each AMD CU. To circumvent this limitation we developed new kernels that reduce both the amount of shared memory used and the number of times the shared memory is accessed. In order to reduce the pressure on the shared memory we developed kernels with 8 of the 12 tensor contractions being performed by the matrix cores on each AMD CU.

The latest CDNA2 architecture includes two new intrinsic matrix instructions for FP64 wavefront based matrix-matrix multiplication, namely `V_MFMA_F64_16x16x4f64` and `V_MFMA_F64_4x4x4f64`. The former multiples a 16x4 matrix with a 4x16 matrix, and the latter performs four separate multiplications of 4×4 matrices with 4×4 matrices. The first variant delivers twice the regular vector instruction throughput, while the second variant achieves at most the same throughput as regular vector instructions. We have found for the BK3 operations that the `V_MFMA_F64_4x4x4f64` performs better in experiments despite its lower nominal throughput

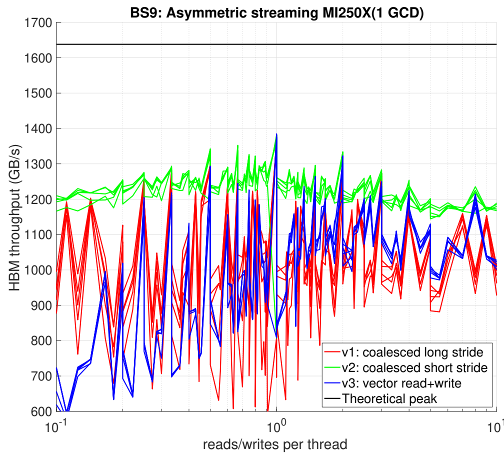


Figure 10: Throughput for three different implementations of the BS9 asymmetric streaming benchmark executing on a single GCD of an AMD MI250X using ROCm 5.2.0.

Figure 11: Comparison of the NVIDIA GA100 streaming multiprocessor and AMD MI200 series CDNA2 core architectures.

which we attribute to allocating less matrix values to each thread.

In Table 7 we show the best obtained device memory throughput and floating point throughput for BK3 in double precision (FP64) and single precision (FP32) over a range of polynomial degrees (N) [1]. In our implementation we pad out the data and operator matrices in size to multiples of 4 to match the dimension restrictions of the MFMA instructions. This means that we are performing extraneous arithmetic operations when the node count or quadrature node count are not multiples of 4 in each direction. However we do not include these extraneous operations in the floating point throughputs reported in Table 7.

For the FP64 results in Table 7 we see that the MFMA-based kernel implementations prove to be more efficient than the kernels that only use vector instructions. In all cases the FP64 kernel memory throughput is close to or exceeds 1TB/s. The FP32 matrix instructions only prove to be more efficient than vector instructions at polynomial degrees $N \geq 12$. Although the vector based kernels degrade in performance as N increases, the matrix instruction version gets close to 1TB/s throughput at $N = 14$. The improvements at $N = 14$ can in part be attributed to the reasonable match of the tensor contraction sizes with the 4×4 MFMA instructions.

[1]For these calculations we used the FP64 (`__builtin_amdgcn_mfma_f64_4x4x4f64`) regardless of whether we are performing calculations on FP32 or FP64 data.

	FP64		FP32	
N	HBM (GB/s)	GFLOPS	HBM (GB/s)	GFLOPS
1	982	1144	996	2271
2	951	1421	937	2724
3	1013	1840	867	3051
4	1026	2192	901	3720
5	988	2426	847	4012
6	1037	2879	992	5295
7	958	2963	961	5708
8	974	3321	924	6046
9	1040	3663	908	6489
10	1033	4205	864	6697
11	1000	4362	830	6927
12	965	4498	791	7078
13	1086	5481	913	8719
14	1018	5716	948	9628

Table 7: Best obtained DEVICE memory throughputs and floating point throughputs for BK3 in double precision (FP64) and single precision (FP32) over a range of polynomial degrees (N). Calculations performed on a single GCD of an AMD MI250X GPU using HIP from ROCm 5.2.0. Results shown in red were obtained using a combination of regular vector and matrix instructions `__builtin_amdgcn_mfma_f64_4x4x4f64` whereas results shown in black were obtained using only regular vector instructions. $N+2$ quadrature points were used in each direction for these experiments.

2.5 Performance tuning for the non-tensor MAGMA backend on AMD GPUs

The non-tensor MAGMA backend performs basis actions using standard matrix-matrix multiplication (GEMM). The matrix dimensions are often very skewed, as shown in Figure 12. The output matrix is often very wide, while all the other dimensions are relatively small. The shape shown in Figure 12 makes it a candidate for *batching*, e.g. by subdividing the output matrix across the columns and launching independent multiplications through batch GEMM. The decision to use batch GEMM depends on many parameters, such as the compute precision, the matrix dimensions, the GPU architecture, and the subdividing dimension nb (shown in Figure 12). Another decision to be taken is whether to choose the vendor routines (i.e. `hipblas`) or MAGMA's own kernels. The selection of the best performing basis action is a challenging task, and may be di cult to maintain on the long term.

The MAGMA team has successfully integrated a lightweight tuning layer, called the *GEMM selector*, for choosing the best performing kernel given a set of certain dimensions. The selector relies on two components, (1) tuning data and (2) a decision maker. The tuning data are generated o ine using benchmark sweeps for all the available GEMM routines in cuBLAS, hipBLAS, and MAGMA, using the typical dimensions often found in libCEED's bake-o problems. Such data are stored in lookup tables, which represent the *knowledge database* for the selector, as it recommends the best candidate kernel for a given dimension. A lookup table is stored for every GPU architecture. The decision maker searches the corresponding lookup table and tries to nd the closest match for the dimensions given at run time. Figure 13 shows the GEMM selector recommendations for the MI250X using double precision. The gure shows that there is no clear winner for all sizes, which proves the value of having a robust tuning layer for the non-tensor backend.

Figures 14 and 15 show the impact of integrating the GEMM selector into the non-tensor backend on the 3D di usion problem. Performance gains are observed in most cases against the original backend, which had hard-coded tuning for the V100 GPU only. The robustness of the GEMM selector on the long term can be maintained by keeping the tuning data up to date. This only requires updating/adding lookup tables, with very minimal changes in the backend itself.

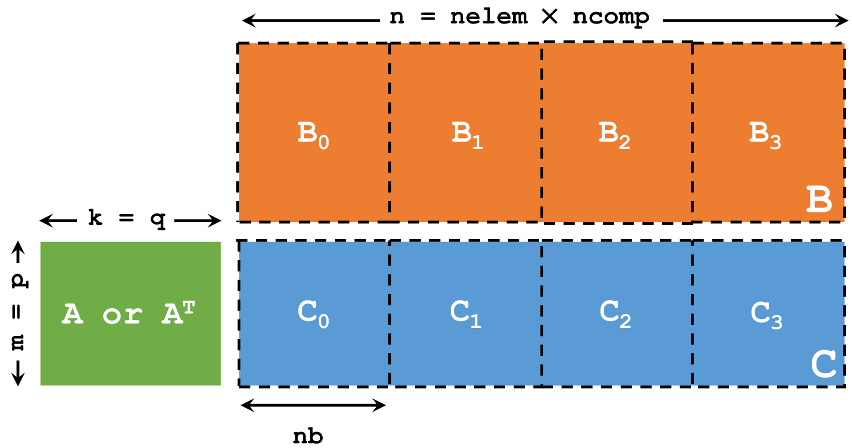


Figure 12: Shape of matrix multiplication for the non-tensor basis actions

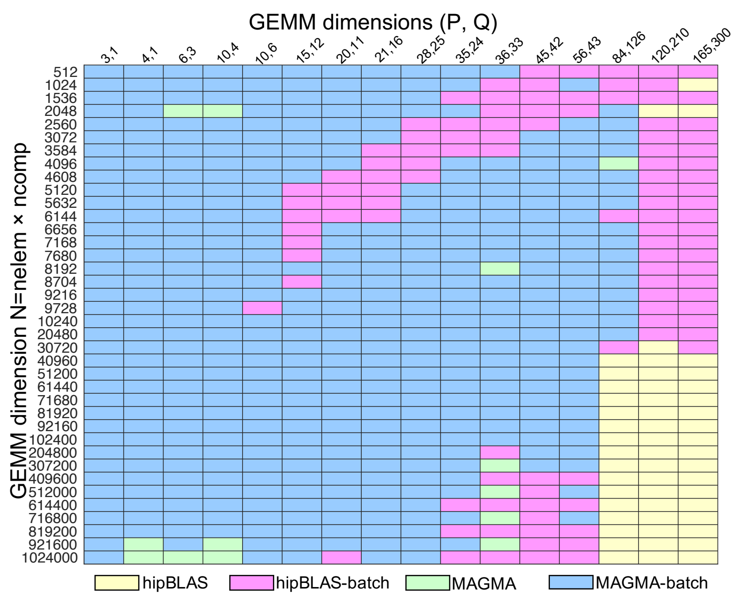


Figure 13: Colormap of the DGEMM selector for the MI250X

2.6 Solid mechanics with matrix-free p-multigrid on Lassen, Summit, Perlmutter and Crusher

In this section we present a study om the use high order nite elements on coarse meshes on real-world structural mechanics problems that have numerous reentrant corners and Dirichlet (xed/clamped) to Neumann (free or applied traction) boundary condition transitions which causes stress singularities. Consider a unit cube with radius 0.3 cylindrical hole, xed to a rigid boundary on one end and with applied tangential traction on the other. Figure 16 shows the deformed state and strain energy function for a Neo-Hookean material with Young's modulus 2.4 and Poisson ratio 0.4, and applied traction of 0.2. We perform a convergence study using linear and high-order geometry meshes produced by Gmsh [19], which can generate arbitrary order curved meshes. Figure 17 shows the relative error in predicted total strain energy (reference value computed on a highly-resolved mesh) versus DoFs for h and p re nement of the 36-element (3 layers deep) mesh evident in Figure 16 (`mesh A`) as well as a more resolved `mesh B`. We observe that p re nement of very coarse meshes is the most e cient path to accuracy.

Next, we show that solve costs decrease with increasing p using matrix-free p-multigrid and thus Figure 17

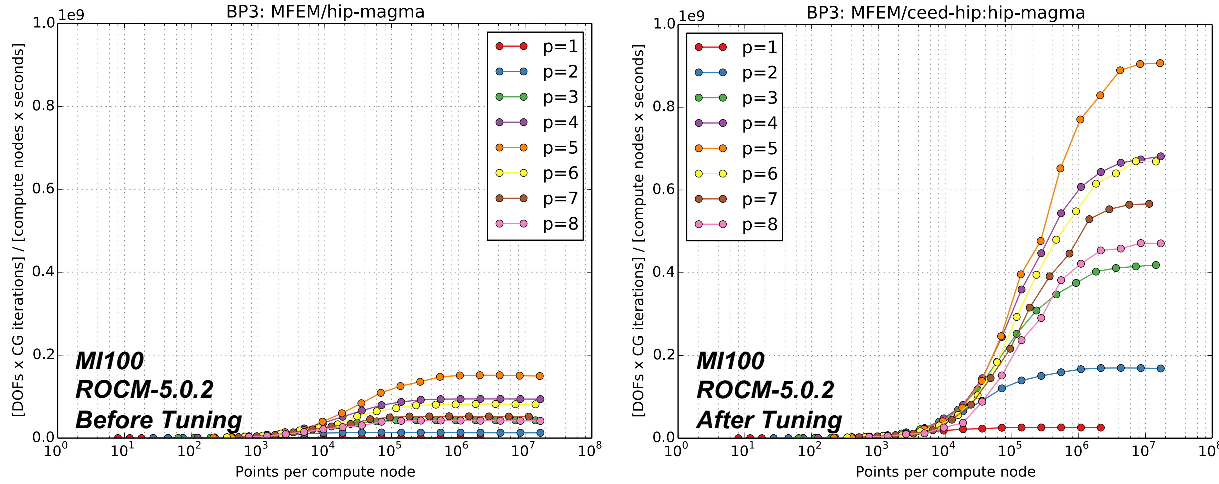


Figure 14: Performance of the 3D diffusion problem using MAGMA's non-tensor backend on the MI100 GPU.

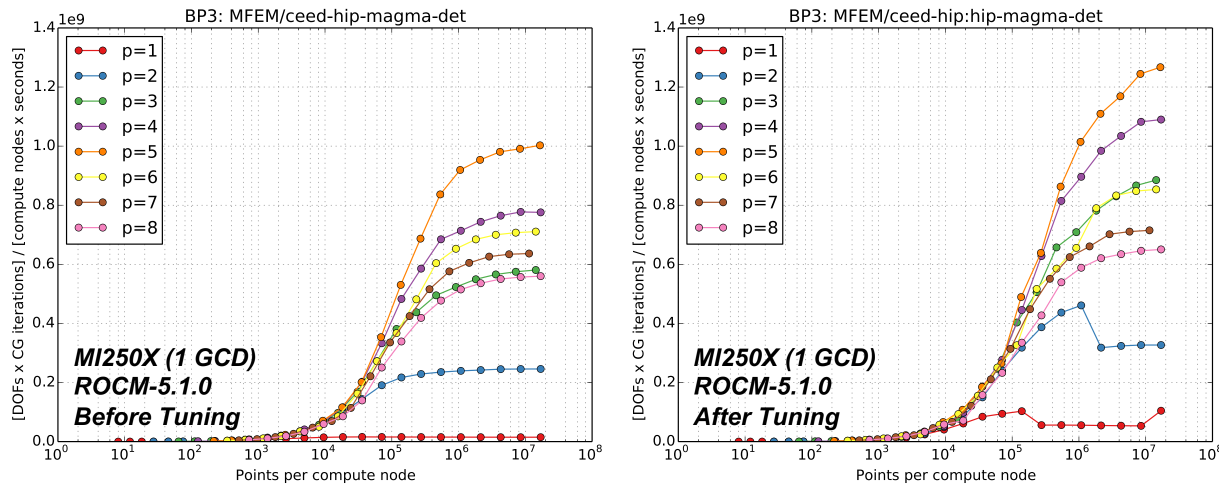


Figure 15: Performance of the 3D diffusion problem using MAGMA's non-tensor backend on the MI250X GPU

is in fact generous to the low-order methods. As in [22], we continue to consider the Schwarz Primitive surface extrusion under load, which exhibits interesting geometric and material nonlinearities. Using a 24-element 2D manifold mesh of a single unit cell embedded in 3D, replicated to the prescribed extent in each embedding dimension. This mesh is partitioned and distributed using ParMETIS, then re ned with new nodes projected to the closest point on the implicit surface

$$\cos 2\ \ x + \cos 2\ \ y + \cos 2\ \ z = 0$$

The resulting manifold mesh is extruded normal to this surface to the prescribed thickness and number of layers. Figure 18 shows such a model loaded to about 12% strain on an extent (8 8 8) model with about 11.8 million DoF (MDoF).

Results are computed on GPU-based environments on LLNL's Lassen, OLCF's Summit and Crusher, and NERSC's Perlmutter. Lassen and Summit are both IBM POWER9 machines with 4 and 6 NVIDIA V100-SXM2 16 GiB GPUs per node, respectively. This study used the open source packages PETSc-3.17 [4], hypre-2.24 [14, 3], Kokkos-3.6 [36], ParMETIS 4.0.3 [20], libCEED-0.10.1 [5], and Ratel-0.1 [6]. The numerical experiments preload by doing a crude tolerance solve that is discarded before starting timers in

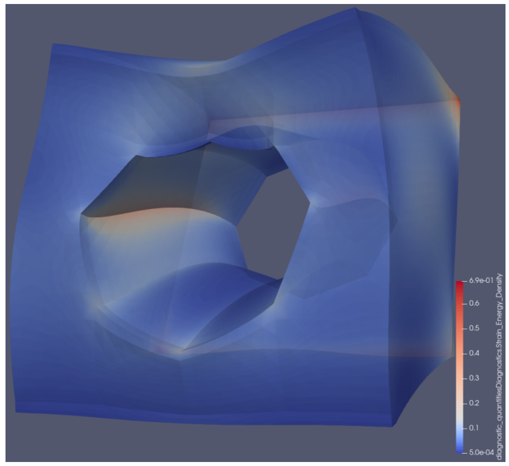


Figure 16: Visualization of the deformed state and strain energy singularities on `mesh A` of Figure 17, refined 3 times by splitting each hexahedron in 8 without snapping to geometry, and solved using Q_2 finite elements. There are physical singularities on the back surface (e.g., top-right corner) and non-physical singularities at the weak reentrant corners of the hole (which do not exist in the smooth model with exact cylinder).

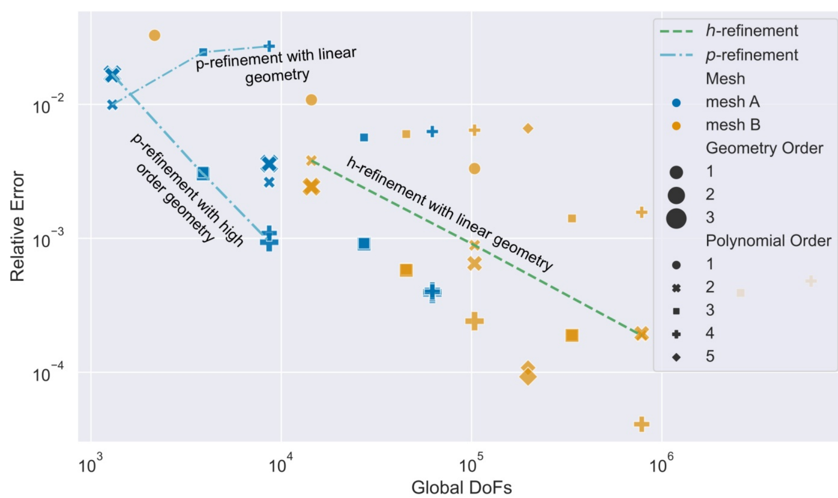


Figure 17: Accuracy study showing relative error in total strain energy Ψ versus DoFs for the bending experiment Figure 16 under both h refinement (same shape) and p refinement (same color) with low and high order geometry. The Pareto front is toward the lower left and we observe that h refinement always moves away from optimality. The slope of h refinement is the same for all meshes and solution orders. p refinement is very efficient so long as the geometry is at least quadratic, but causes errors to increase when p refining on linear geometry due to resolution of the non-physical singularities.

order to provide consistent timing representative of longer-running simulations. For more accurate pro ling of individual events, the pro led runs include some unnecessary synchronization with the GPU, introducing a slight latency penalty to the smallest model sizes.

We compare e ciency of di erent machines and di erent parallel scale using the model from Figure 18 with (nondimensionalized) parameters thickness 0.2, Young's modulus 1, and Poisson ratio 0.3, xed to the left wall with a compressive traction of 0.02 applied from the right. This model requires 5 to 7 Newton

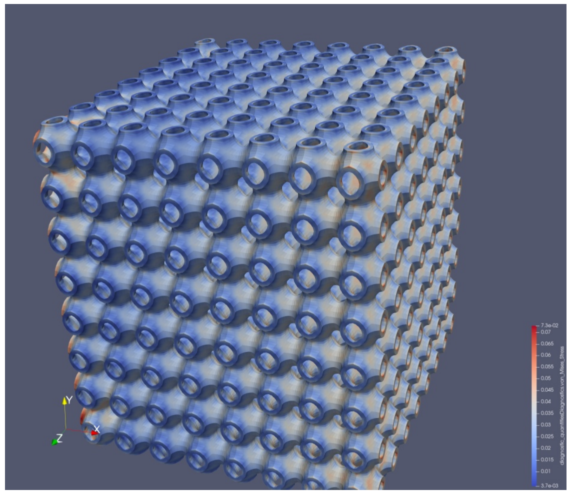


Figure 18: Extruded Schwarz Primitive surface under 12% compressive strain, colored by von Mises stress. The left wall is fixed and a compressive force is applied to the facing surfaces on the right. The simulation used 2 refinements, 2 layers, thickness 0.2, and Q_2 elements.

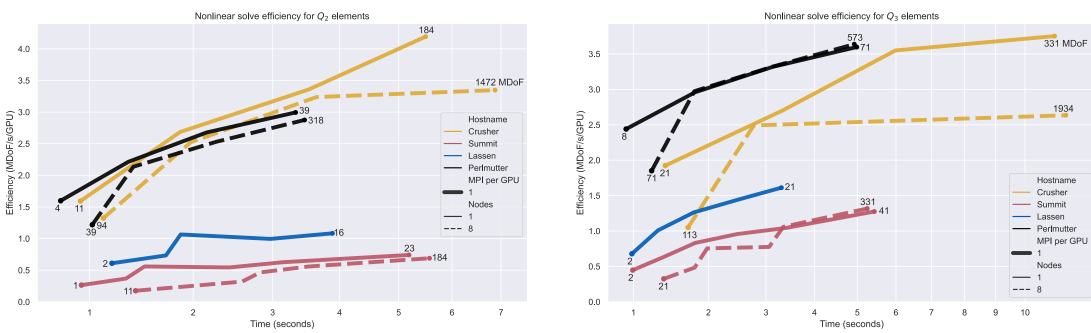


Figure 19: Efficiency per Newton iteration versus time for Q_2 (**left**) and Q_3 (**right**) finite elements using matrix-free Newton-Krylov with p-MG preconditioning and BoomerAMG coarse solve. Problem sizes (in MDoF) are annotated for the minimum and maximum sizes for each host and number of nodes combination. The impact of latency is ever-present, with memory capacity limiting the right end of each curve. Ideal weak scaling is evident for Q_3 on Perlmutter for Newton step time above 1.8 s where the 1-node and 8-node curves coincide, while communication latency leads to degradation at the smallest problem sizes (2 MDoF/GPU with time around 1 s).

iterations across the range of resolutions, with each linear solve needing 9 to 25 preconditioned CG iterations to converge to a relative tolerance of 10^{-3} in the natural norm, with CG condition number estimates from 9.5 to 61 (mostly less than 15 iterations and condition numbers less than 20; depending on the Newton step).

We sweep through a range of Primitive model extents up to 20^3 per node of Crusher (184 MDoF), solve each model, and plot e ciency versus time per Newton iteration for Q_2 and Q_3 elements in Figure 19. The Q_2 model is as depicted in Figure 18 with 2 re nements and 2 extruded layers, while the Q_3 model uses only one extruded layer to achieve somewhat better accuracy. In such plots, perfect weak scaling would have the 1-node and 8-node curves on top of each other, with strong scaling limits visible in the minimum time at which acceptable e ciency can be achieved. This human-centric gure is meant to assist the analyst with cloud or HPC access in choosing an e ciency-versus-time tradeo . For example, one may look at the right portion Figure 19 and decide that under 2 s per Newton iteration (about 10 s) for the total nonlinear solve) delivers an acceptable e ciency time tradeo . Examining the Perlmutter curve with about 3 MDoF/s/GPU at 2 s, the target problem would be scaled to about 6 MDoF/GPU. The 1-node and 8-node Perlmutter curves lie on top of each other here, indicating that one can solve a 24 MDoF problem on one node (4 GPUs) with the same efficiency as a 192 MDoF problem on 8 nodes.

Note that AMG requires a deeper V-cycle for the larger problem size, but this latency impact is hidden at the 2 s solve time with Q_3 elements. Compare with left of Figure 19, in which there is a slight efficiency penalty to the weak scaling since a greater fraction of the solve time is spent in AMG when using Q_2 elements. The solve can be made somewhat faster by using more GPUs (with some drop in efficiency) or more efficient by using fewer GPUs (while needing to wait longer). Different architectures can readily be compared in this metric by normalizing the efficiency axis by energy (DoF/joule) or monetary (DoF/dollar) cost.

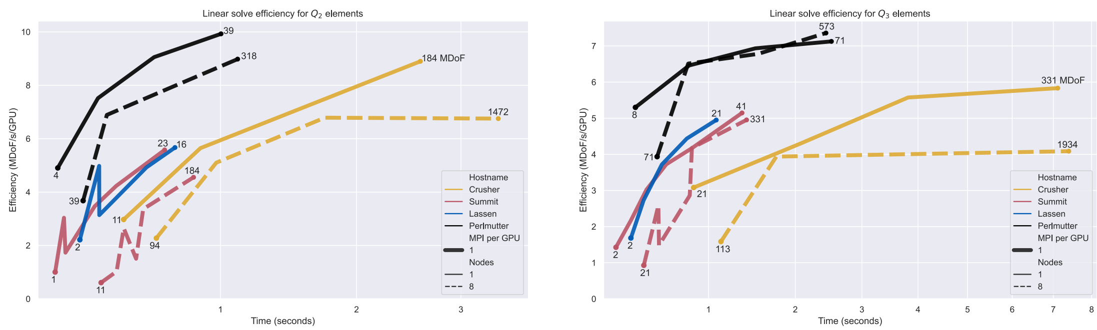


Figure 20: Linear solve efficiency spectrum for Q_2 (**left**) and Q_3 (**right**) finite elements using matrix-free Newton-Krylov with p-MG preconditioning and BoomerAMG coarse solve. The times and efficiencies are per Newton iteration. Problem sizes (in MDoF) are annotated for the minimum and maximum sizes for each host and number of nodes combination.

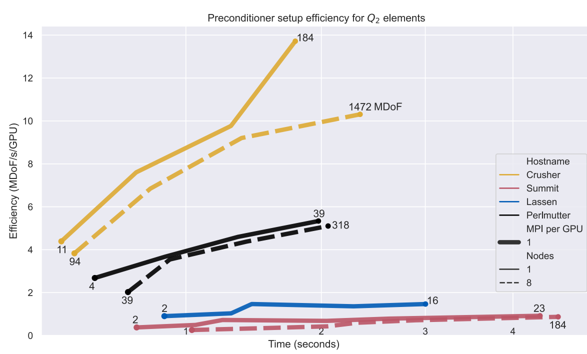


Figure 21: Preconditioner setup efficiency spectrum for Q_2 finite elements using matrix-free Newton-Krylov with p-MG preconditioning and BoomerAMG coarse solve. The times and efficiencies are per Newton iteration. Problem sizes (in MDoF) are annotated for the minimum and maximum sizes for each host and number of nodes combination.

The linear solve in Figure 20 is communication-intensive since each preconditioner application goes through a V-cycle (with an increasing number of levels as the model gets larger). Figure 21 considers preconditioner setup, which consists of algebraic multigrid analysis and Galerkin products as well as a few Krylov iterations to calibrate the smoothers. Assembly of the coarse Jacobian (`SNESJacobianEval`) exhibits nearly perfect problem-size independence at about 20-25 MDoF/s/GPU (with Q_2 elements) on Crusher and Perlmutter, and is thus always less than 20% overhead. The relative cost of Jacobian assembly and preconditioner setup both decrease when going to Q_3 elements because the coarse problem is a smaller fraction of the fine problem size.

2.7 Mixed-precision libCEED kernels and compiler insights for AMD MI250X

In CEED-MS38, Section 5.4, we compared single- and mixed-precision performance to that of double precision for libCEED operators on NVIDIA (V100) and AMD (MI100) GPUs [22]. Those initial MI100 results were surprising, particularly for the MAGMA backend (Figure 62). Because the MAGMA backend uses separate kernels for the suboperations in libCEED's operator application sequence, $\mathcal{E}^T B^T D B \mathcal{E}$, the mixed-precision mode used was what we termed low-high, meaning the input/output vectors to each kernel are in the lower precision (single), while all computation is performed in higher precision (double). All intermediate data storage E- and Q-Vectors in libCEED's terminology are thus in single precision, which lowers the total data movement to and from main memory. However, as seen in the previous report, there were several curiosities: first, that the all-float performance would often jump above 2× speedup over double, and second, that the mixed-precision case enjoyed almost no speedup at all, particularly for higher orders of basis functions, despite these kernels being memory bound. Both of these trends also did not match the CUDA results for an NVIDIA V100 GPU.

Recent developments in the MAGMA backend and further investigations into the performance of these kernels have yielded some answers to these questions. Most importantly from a performance point of view, the MAGMA backend was updated to use runtime compilation via `nvrtc`/`hiprtc` for its tensor basis kernels. This was not done with performance in mind, but rather more general code maintainability benefits: it makes the MAGMA backend more similar in structure to other GPU backends, it adds more coverage to the parameter ranges available through the MAGMA backend, and it greatly reduces the time to compile the library itself. However, we discovered that it also seemed to improve the performance of the MAGMA backend on AMD GPUs, both MI100 and MI250X. We found that a difference in flag handling between `hipcc` and `nvcc` appeared to be responsible: the same optimization flag was used for both host and device code with `hipcc`, which was not the case when building the CUDA version of the MAGMA backend with `nvcc`. This meant the default libCEED CPU optimization flag which was set to `O` was being used for ahead-of-time HIP kernel compilation. The runtime-compiled kernels, on the other hand, were using the `hipcc` default of `O3` in the absence of setting a specific level. Because `nvcc` doesn't propagate the optimization flag through to device code, this disparity between ahead-of-time and runtime compilation of the kernels wasn't seen in the CUDA MAGMA backend.

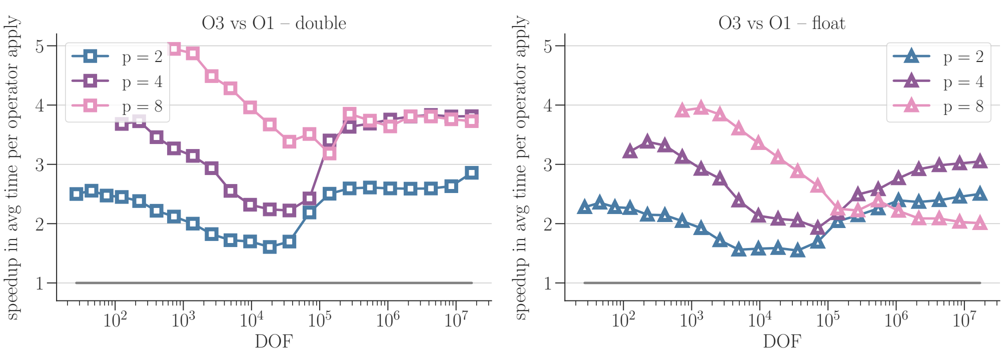


Figure 22: Comparison of 3D diffusion operators compiled with `O3` versus `O1` optimization levels for the `magma-det` backend, in double precision (left) and single precision (right). Results obtained with ROCm 5.2 on one GCD of an AMD MI250X GPU.

In Figure 22, we show the speedup in the average time to apply the three-dimensional diffusion operator when using `O3` versus `O1`. All results here were obtained via runtime compilation, but the `O1` results are a good proxy for older, ahead-of-time-compiled MAGMA basis kernels used in previous reports, as explained above. For these tests, the operator was applied 2000 times, followed by one device synchronization call to ensure all kernels had completed, then the total time was divided by the number of applications. The speedup is large often between 2× and 4× for `O3`.

Now if we reconsider the mixed-precision MI100 experiments from CEED-MS38 on the MI250X, we see that using `O1` results in similar (unexpected) trends of poor mixed-precision performance despite excellent single-precision performance, relative to double. See the left side of Figure 23; compare with the right side of Figure 62 in CEED-MS38. However, on the right side of Figure 23, we see the same tests with `O3` (the default flag for `hiprtc`). From Figure 22 we know that the double and float operators in the right plots are faster than their counterparts on the left, but we also see that for second and fourth order basis functions, the "low-high" mixed-precision case retains most of the speedup of all-float, as we would hope for memory bound kernels. The all-float cases also no longer spike above 2× speedup. When we move to eighth order basis functions, however, a new result appears: the mixed-precision case is actually *better* than all-float.

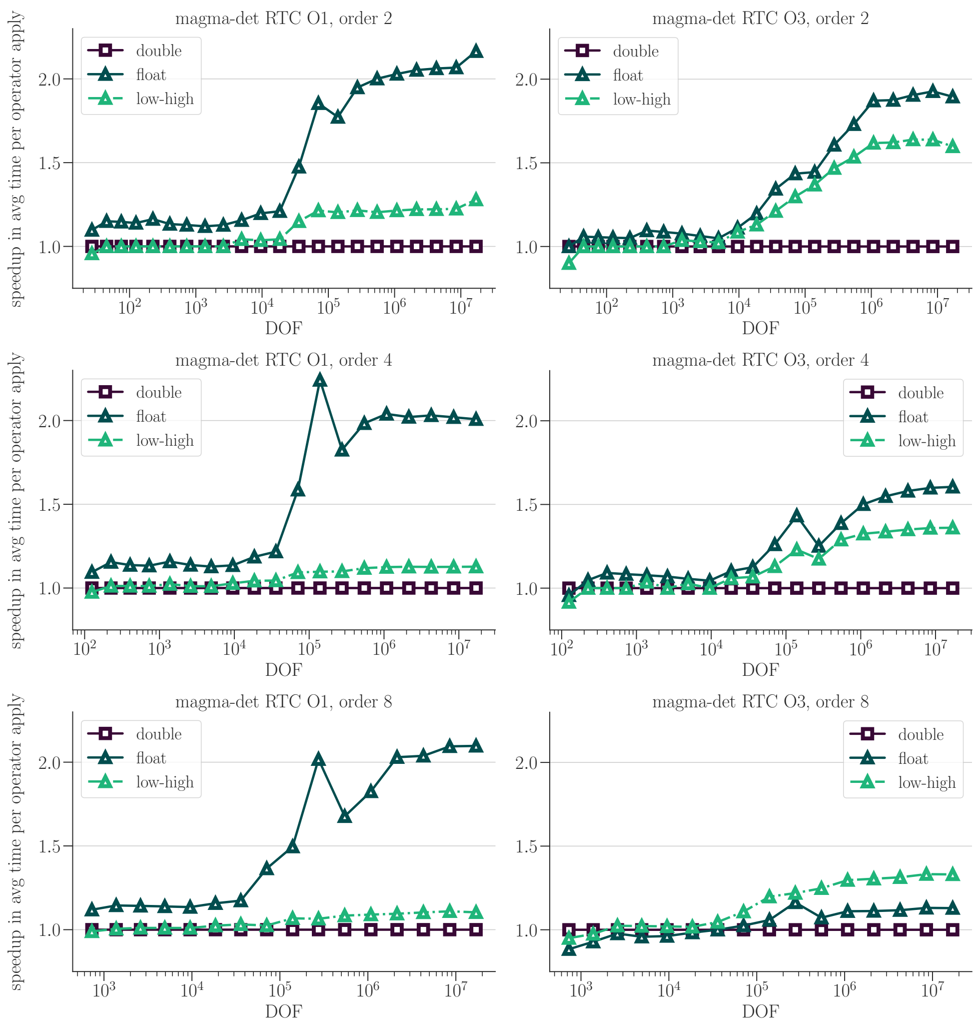


Figure 23: Comparison of mixed-precision and single-precision performance versus baseline double precision for the HIP version of the `magma-det` backend using runtime compilation via `hiprtc`, with either `O1` or `O3` optimization levels. Tensor-product basis functions of order 2, 4, and 8 are considered for the 3D diffusion (Poisson) operator. `O1` performance approximates previous tests of the MAGMA backend for HIP, which did not use runtime compilation; see MS38 (Figure 62). Results obtained with ROCm 5.2 on one GCD of an AMD MI250X GPU.

Through investigations with `rocprof` and comparing the output of the `hiprtc` compiler (as reported by using the `--save-temps` flag), we can perhaps explain these results. In Table 8, we collect some metrics and information from the compiler output for a sample computational kernel: the non-transpose gradient basis kernel, used in the benchmark results shown above. The top portion of the table contains four metrics: the number of scalar registers used by the kernel, the number of vector registers, the private scratch memory usage per thread (in bytes), and the occupancy. The key points for comparison between the optimization levels are that `O3` increases register usage, which sometimes drops the occupancy, but it also reduces the scratch memory usage to zero. The scratch memory usage seems to be important, as demonstrated in Figure 24, where we see that the `O1` kernels (left) sometimes report less than 50% total read/write of double precision, while the mixed-precision kernels are often closer to double not the goal of the low-high scheme. For the `O3` kernels (right), that is not the case. The total memory movement for each precision is also reduced with `O3`. (Note that Figure 24 combines all kernels in the operator, not just the non-transpose gradient basis kernel highlighted in Table 8.) Along with loop unrolling done with `O3` but not `O1`, the lowered memory movement is a likely cause of improved performance for these memory-bound kernels.

		$p = 2$			$p = 8$		
		double	lo-hi	oat	double	lo-hi	oat
		`O1` / `O3`	`O1` / `O3`	`O1` / `O3`	`O1` / `O3`	`O1` / `O3`	`O1` / `O3`
kernel info	Num scalar regs	22 / 16	22 / 16	24 / 16	22 / 16	22 / 16	24 / 16
	Num vector regs	16 / 45	16 / 45	18 / 32	18 / 98	18 / 98	18 / 70
	Scratch mem	80 / 0	80 / 0	48 / 0	176 / 0	176 / 0	112 / 0
	Occupancy	8 / 8	8 / 8	8 / 8	8 /4	8 / 4	8 / 7
*num. of instructions**	Packed FP32 ops†			0 / 42			0 / 315
	`v_mov_b32`	33 / 23	33 / 23	53 / 91	34 / 63	34 / 63	55 / 545

Table 8: Selected HIP compiler output information for `magma_gradn_3d_kernel` – 3D non-transpose gradient basis kernel from the MAGMA backend.
*Instructions as counted by number of kernel ISA lines where a relevant instruction was found; any branching/jumps ignored.
†Combined total of `v_pk_add_f32`, `v_pk_mul_f32`, and `v_pk_fma_f32`.

Next, we turn our attention to the loss of performance for oat with $p = 8$. The second part of Table 8 contains information on the number of times certain instructions appear in the generated ISA of the kernel: rst, the packed oat computational instructions, only available on MI200; and second, move instructions. With `O1`, the compiler does not emit any packed FLOP instructions; with `O3`, it does, but it also increases move instructions, as it needs to move data to align it properly for packed instruction use. Here we also see the di erence between $p = 2$ and $p = 8$: not even 2x more move instructions for `O3` versus `O1` with $p = 2$, but nearly 10× for $p = 8$! (This analysis does ignore any possible branches or jumps in the assembly code, and merely counts the number of lines containing the instructions.) It is likely that in this case, the extra move instructions required to use the packed FLOP instructions render any computational bene ts moot, which is why the mixed-precision case which cannot even try to use packed instructions as all computation is happening in double outperforms all- oat for this case.

3. CEED PERFORMANCE IMPROVEMENTS FOR AURORA

3.1 Extending the MAGMA portability with OneAPI

The MAGMA library has been originally designed and implemented for heterogeneous systems that use NVIDIA GPUs. Support has been extended to OpenCL, Intel Xeon Phi, and more recently to AMD GPUs by translating CUDA code to HIP. To provide support for Aurora, and Intel GPUs in general, we started to port MAGMA to SYCL/DPC++. DPC++ is the direct programming language of the oneAPI programming model interface, providing portability to all supported hardware, including scalar, vector, spatial, and matrix architectures such as CPUs, GPUs, Field-Programmable Gate Arrays (FPGAs), and other accelerators. Thus, of interest for this period was to evaluate the ease of portability, as well as the functional and performance portability of a large numerical library like MAGMA. To do the evaluation we set di erent tests for multicore CPUs, NVIDIA GPUs, and Intel GPUs.

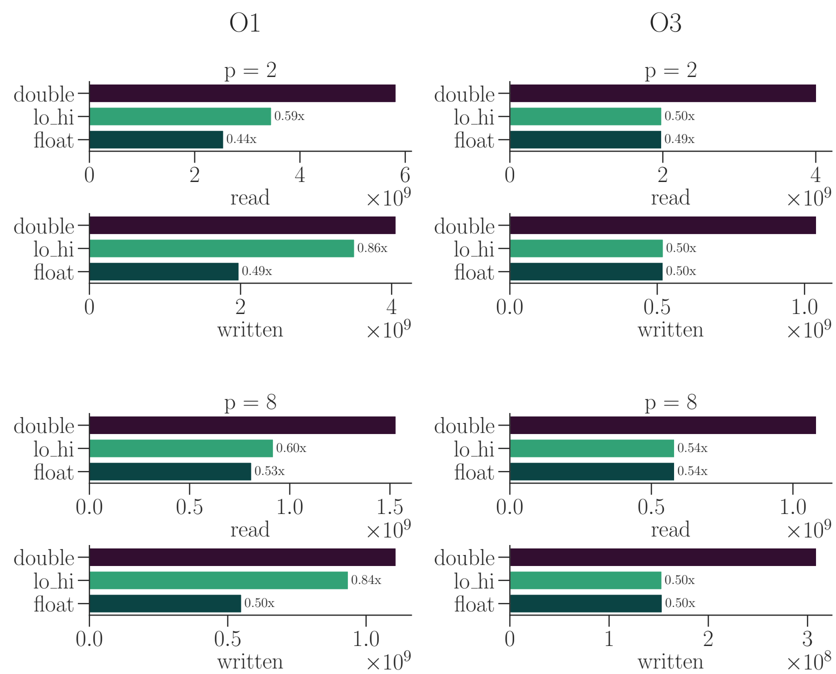


Figure 24: Comparison of data read/write, in bytes, as reported by `rocprof` for double, single, and mixed-precision 3D diffusion operator applications in the `magma-det` backend for tensor product basis functions. The profiled cases use the same number of total DOF in the solution (4,276,737), for second and eighth order basis functions. Labels for mixed and float indicate the fraction of the double-precision value. Values are averaged over 4 operator applications.

The GEMM design and implementation are of fundamental importance in HPC as many scientific computing applications, including the main kernels in CEED, are designed in terms of GEMM operations. GEMM is expected to run close to machine peak of the underlying hardware so that applications using GEMMs derive their high-performance and performance portability across different hardware from highly efficient GEMM implementations. This is the reason that we concentrated on the GEMM kernel for this evaluation. Performance portable GEMM implementations however are very challenging to develop even for just one specific architecture, let alone having a single code to be performance-portable across architectures. Still, our goal is to evaluate exactly that, i.e., to what extent can a single DPC++ GEMM implementation be performance-portable.

The DPC++ Compatibility Tool (DPCT) in Intel's oneAPI was used successfully for an initial port of selected MAGMA kernels to DPC++, followed by manual changes. In particular, the DPCT tool was used to recursively migrate the SGEMM files and headers (about 50 files, including all magma header files and GEMM sources). A motivation to choose FP32 is that FP32 arithmetic is often more than 2× faster than FP64 for many GPUs and many applications, including machine learning and deep neural networks, are currently aiming to leverage this performance boost by reducing the accuracy (even to FP16) and possibly regaining it with mixed-precision calculations where the bulk of the computation is in reduced FP16 or FP32 precision.

Figure 25, Left shows the performance of the migrated DPC++(MAGMA) SGEMM algorithm on an AMD multicore CPU system. It outperforms significantly the OpenMP C++(OpenMP) implementation, and even the MKL implementation [18]. This result is significant as it shows that an algorithm designed for NVIDIA GPUs can be very efficient for multicore CPUs as well. Note that the DPC++(MAGMA) implementation outperforms even MKL. This means parallelization, blocking for reduced communication, and vectorization have been all efficiently achieved. Thus, since a fast multicore GEMM is very challenging to develop, this is one illustration of both functional and performance portability of MAGMA GEMM, and

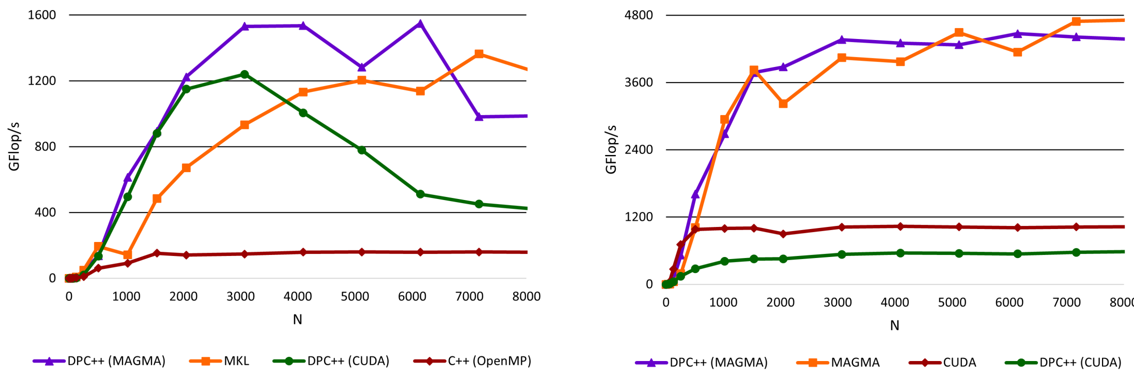


Figure 25: Left: Performance of MAGMA's SGEMM translated to DPC++ and benchmarked on multicore AMD EPYC 7742 64-Core 2.25 GHZ CPU. Right: Performance of MAGMA's SGEMM translated to DPC++ and benchmarked on NVIDIA GeForce RTX 3060 GPU.

arguably the entire MAGMA because of the GEMM importance, to multicore CPUs using the Intel's oneAPI programming model and toolkit. Furthermore, MAGMA never had a port to just multicore CPUs, and this result shows that the oneAPI port as being developed is a feasible solution to easily provide this functionality in a performance portable way.

Figure 25, Right shows the performance of the migrated MAGMA DPC++(MAGMA) code, originally tuned for NVIDIA GPUs, to an NVIDIA GPU system. The migrated code retained the performance of the CUDA implementation. This is signi cant result as it illustrates that DPC++ is expressive enough for parallel algorithms that have to map and run well on NVIDIA GPUs, as it matches in performance CUDA that is designed for NVIDIA GPUs. Also, it shows that the Intel oneAPI compiler is very good in generating highly optimized code for NVIDIA GPUs. This is the ideal outcome for a new language, its compiler, and a translation tool   the tool to translate a highly-optimized code to a new language, compile the new code, and achieve the same performance as the original code on hardware for which the original code had been tuned.

Combined with the results for multicore CPUs, we conclude that after a full translation of MAGMA is complete, the same code can be used for both functional and performance portability to NVIDIA GPUs, multicore CPUs, and Intel GPUs.

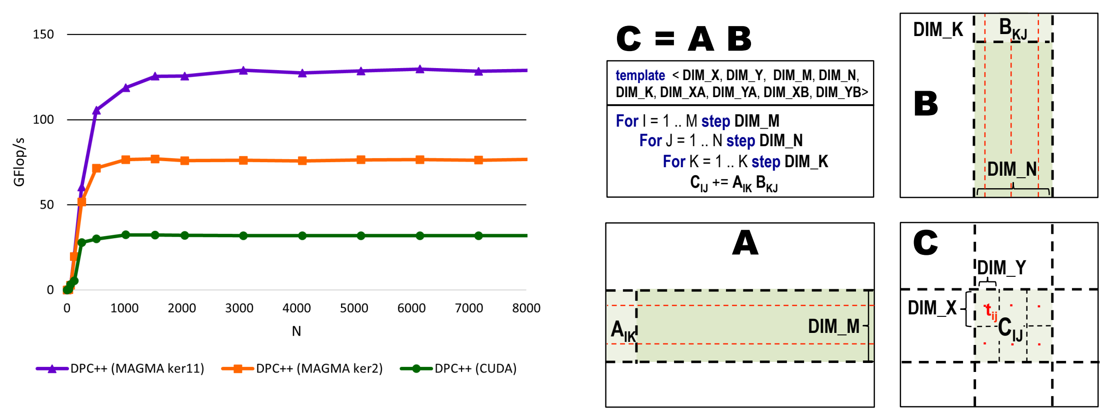


Figure 26: Left: Performance MAGMA SGEMM versions translated to DPC++ and benchmarked on Intel UHD Graphics P630 GPU; Right: MAGMA's templated GEMM producing various versions (e.g., MAGMA ker2 and MAGMA ker11) through autotuning nine templated parameters.

Initial porting results, e.g., for the migrated DPC++(MAGMA) code, were very poor for Intel GPUs. The MAGMA DPC++ SGEMM algorithm never exceeded two GFlop/s in performance for the Intel UHD

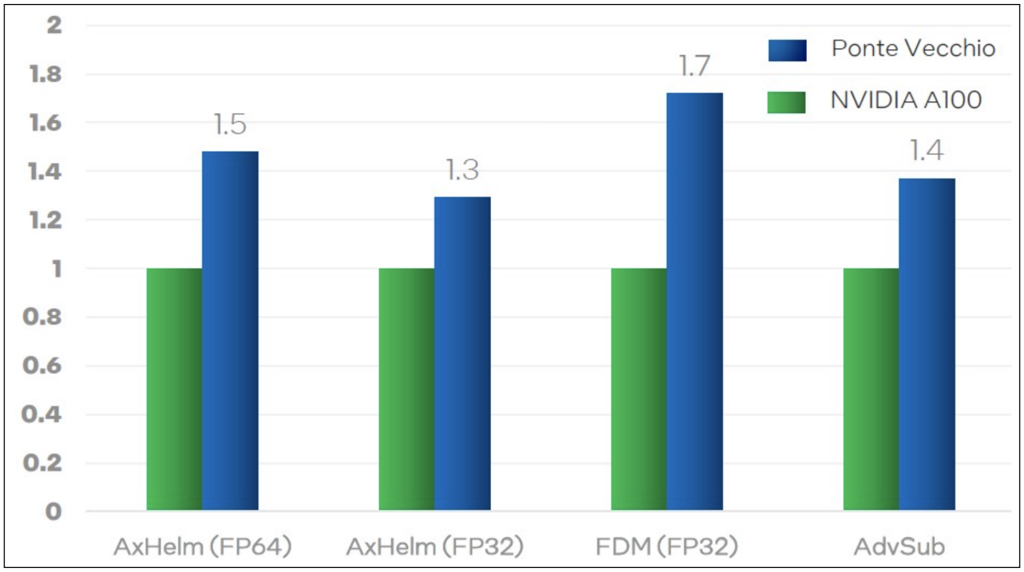


Figure 27: Relative performance of NekRS benchmarks with problem size of 8196 (Averaged throughput, higher is better).

Graphics P630 GPU. This indicated that the parameters originally used in the MAGMA SGEMM algorithm, needed to be updated to accommodate for the Intel GPU architecture. Multiple constant sets were tested next, to discover ker2 and ker11, which increased the performance to more than ten times that of the standard SGEMM algorithm, as shown in Figure 26, Left. Optimal algorithm parameters are yet to be determined. Without speci c details about the Intel GPU architecture, several hundred parameter combinations must be tested through trial and error (following an autotuning approach). The MAGMA GEMM algorithm is shown in Figure 26, Right. This algorithm is used for GEMM of any type, including single precision. There are nine important constants that tune the algorithm to a hardware architecture.

Ideally, more hardware con gurations will be of interest to test, tune for, and use, and we have ran on some, including high-end Intel GPUs in the early-access precursors of the Aurora system, to nd that the conclusions and main massage of this work remain the same [18].

3.2 NekRS on Ponte Vecchio with Intel OneAPI DPC++ implementation

NekRS has been ported on to Intel's Ponte Vecchio (PVC) GPU system by Kris Rowe (ALCF) in collaboration with Intel sta . In particular, the performance gains in timing results for NekRS benchmarks on PVC are featured in this article (`https://www.servethehome.com/intel-ponte-vecchio-xmx-with-oneapi-detailed-at-hc34/`) that are shown in Figure 27. The results demonstrate the relative performance of NekRS's benchmarks using the problem size of 8196 on a PVC B4 node with averaged gains of 1.5× for `axhelm (FP64)`, 1.3× for `axhelm (FP32)`, 1.7× for `FDM (FP32)`, and 1.4× for `advSub (FP64)`.

The team is continuing to conduct performance analysis both at the kernel level and for the full code with variation of polynomial orders, element counts and environment settings (MPI ranks, number of tiles). The Intel and ANL groups have been holding bi-weekly meetings with the CEED NekRS team to investigate overall performance concerns in detail. The overall results are very promising for the port of NekRS to Aurora.

4. ADDITIONAL TOPICS

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

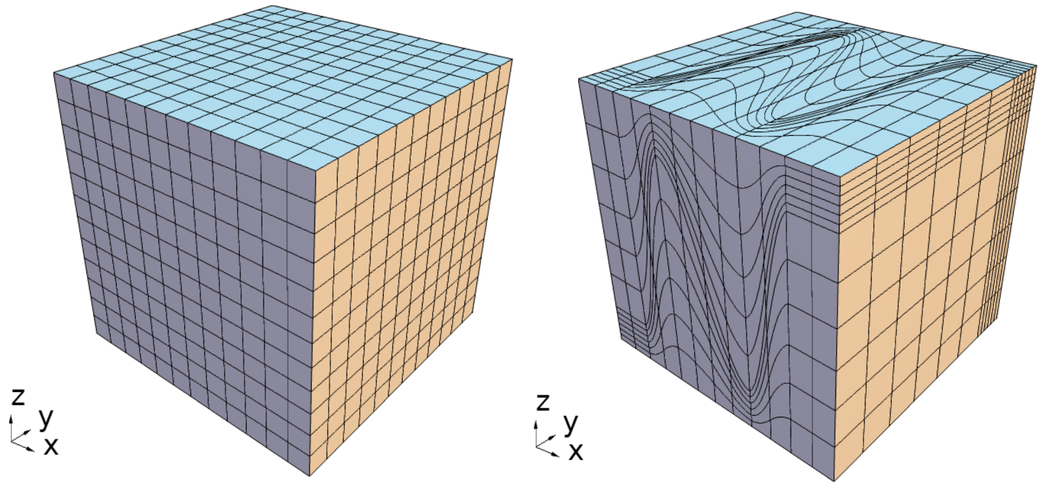


Figure 28: Kershaw meshes: (left) Initial mesh, (right) Mesh deformed using Kershaw deformation parameters $\varepsilon_y = \varepsilon_z = 0.3$.

4.1 Partial assembly for high-order mesh optimization in MFEM

The high-order mesh adaptivity framework in MFEM is based on the Target-Matrix Optimization Paradigm (TMOP) [12]. This framework relies on node movement for minimizing a non-linear objective function that depends on the current and target (desired) geometric parameters: *aspect-ratio*, *size*, *skew*, and *orientation* of each element in the mesh. Prior work has demonstrated the e ectiveness of this framework for adapting high-order meshes to the partial di erential equation of interest and improving the accuracy and computational cost of the solution [11, 13]. The TMOP-based objective function is minimized in MFEM using the Newton's method, which requires the gradient and Hessian associated with the objective function. Original implementation in MFEM relied on a fully assembled Hessian operator for each Newton iteration, and numerical experiments demonstrate that the construction and action of this Hessian operator accounts for majority of the time in mesh adaptivity on CPUs.

Recently, we have improved the mesh optimization algorithms in MFEM to use partial-assembly for the Hessian operator as well as the gradient of the objective function. The assembly of the global sparse matrix (for the Hessian) and vector (for the gradient) has been replaced with pre-computation and storage of data at quadrature points in each element, which is used to perform operator action element-by-element using tensor contractions. We have also developed the capability to perform Jacobi preconditioning for the Hessian, as the diagonal of the operator can be computed through tensor contractions without having the global matrix.

To assess the impact of partial assembly on mesh optimization, we use a mesh deformed using the Kershaw transformation [21] and compare the total time it takes to optimize the mesh on CPUs and GPUs. Figure 28 shows the original $24 \times 24 \times 24$ mesh and the mesh deformed using Kershaw parameters ($_y =$ $_z = 0\;3$). Appendix A of [7] provides the source code used to implement this transformation.

The numerical experiments discussed here were done on Lassen, a Livermore Computing supercomputer, that has IBM Power9 CPUs (792 nodes with 44 CPU cores per node) and NVIDIA V100 GPUs (4 per node). All reported computations utilize a single machine node. The CPU runs use 36 CPU cores, while the GPU runs utilize 4 CPU cores with 1 GPU per core. Table 9 compares the unique degrees of freedom and total quadrature points per core for optimizing meshes of di erent polynomial orders ($p = 1$ 4). The quadrature order is uniformly set to 8, and a shape metric is used to improve the skewness and aspect-ratio of elements in the mesh. In each case, TMOP-based optimization improves the high-order deformed meshes back to the original undeformed state. The reader is referred to [7] for additional details of the problem setup, which we skip here for brevity. Table 9 also shows that the improvements made in the current work lead to a 30-40× speed up on GPUs in comparison to CPUs.

We also provide throughput results that demonstrate how well the proposed algorithms utilize the machine resources as the problem size increases. The plots on Figure 29 show the throughput for the action of the

	Unique Degrees of Freedom			
	$p=1$	$p=2$	$p=3$	$p=4$
	46,875	352,947	1,167,051	2,738,019
	Quadrature points per core ($\frac{E(Q+1)^d}{P}$)			
CPU	279,936			
GPU	2,519,424			

(a)

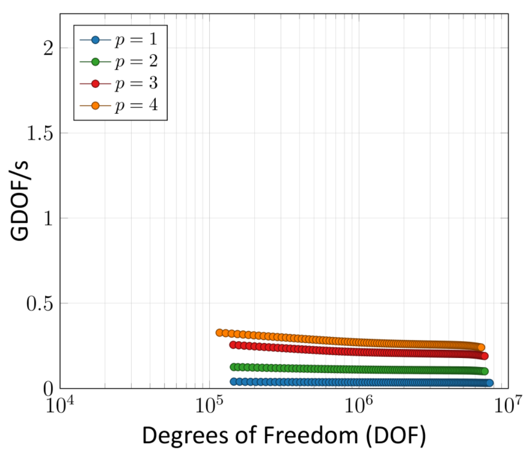
	Time to solution (sec)			
	$p=1$	$p=2$	$p=3$	$p=4$
CPU	18.8	43.0	129.6	224.3
GPU	0.4	1.0	3.9	7.5
	Speedup (GPU vs CPU)			
	47×	**43**×	**33**×	**30**×

(b)

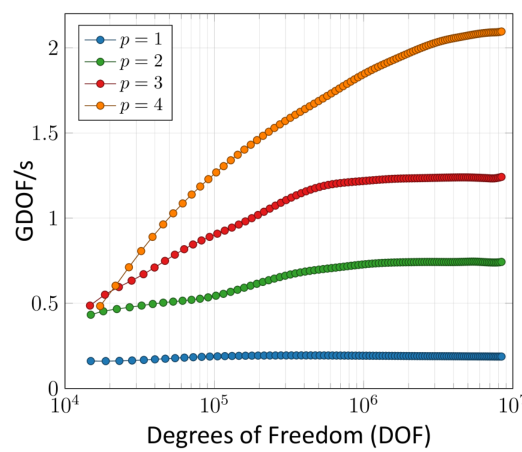
Table 9: For meshes of different orders (p), we compare (a) unique degrees of freedom and total quadrature points per core and (b) total time to solution.

Hessian operator, which is the most cost-intensive kernel in mesh optimization. The throughput is provided in terms of billions of degrees of freedom processed per second vs. the problem size in the CPU and GPU cases. Such plots are useful in comparing the performance of di erent orders and problem size in both the weak and scale limits, see e.g. [23]. For example, higher throughput means faster run time, and from Figure 29 we can see that on both platforms higher orders perform better, but the di erence is much more signi cant on GPUs. Furthermore, while CPU performance is relatively at across problem sizes, the GPU requires signi cant number of unknowns (in the millions of degrees of freedom) to achieve the best results.

Figure 30 shows the throughput for the complete TMOP algorithm for a single GPU. We observe that the higher orders achieve better computational intensity, especially for larger problems. Every data point in Figures 29 and 30 is obtained by timing the computation of a single Newton iteration on the initial deformed mesh. The number of quadrature points per element is set to 3^3 4^3 5^3, and 6^3 for mesh orders 1 2 3, and 4, respectively.



a. 40 IBM Power9 cores



b. 4 NVIDIA Tesla V100-SXM2

Figure 29: Throughput comparison - action of the second derivative operator. Single Lassen node throughput (in GDOF/s, i.e. billions of degrees of freedom per second) of (a) 40 IBM Power9 CPU cores and (b) 4 NVIDIA Tesla V100-SXM GPU.

4.2 Solver advances for high-order discretizations

We have made progress in improving solution times for spectral element and high-order FEM Poisson solves using p-multigrid (pMG) with Chebyshev-accelerated Schwarz smoothing. In a recent article, Lottes [26] demonstrated the bene ts of using 4th-kind Chebyshev polynomials for multigrid smoothing. A particular feature of Chebyshev smoothing is that it is better able to suppress low-wavenumber error than standard

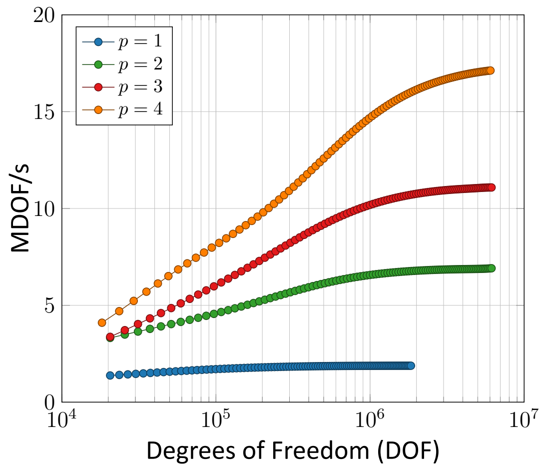


Figure 30: Throughput of the complete TMOP computation. Single NVIDIA Tesla V100-SXM GPU throughput (in MDOF/s, i.e. millions of processed degrees of freedom per second).

Case	Fastest Solver	T_S	Iterations	$\frac{T_D}{T_S}$	$\frac{(T_{crs})_D}{(T_{crs})_S}$
Kershaw ($\varepsilon = 1$)	1^{st}-Cheb, λ_{min}^{opt}, RAS(2,2)	0.09	8	1.75	1.13
Kershaw ($\varepsilon = 0.3$)	1^{st}-Cheb, λ_{min}^{opt}, RAS(5,5)	0.67	28	1.35	1.79
Kershaw ($\varepsilon = 0.05$)	4^{th}_{opt}-Cheb, RAS(12,0)	2.40	88	1.75	2.31
67 pebble	4^{th}_{opt}-Cheb, RAS(12,0)	0.37	12.5	1.81	2.41
146 pebble	4^{th}_{opt}-Cheb, RAS(4,4)	0.15	5.3	1.17	1.21
1568 pebble	4^{th}-Cheb, ASM(12,0)	0.14	3	1.27	2.13

Table 10: Solver configuration with the fastest time to solution.

smoothing strategies. By reducing the number of outer GMRES iterations, these gains in pMG take pressure o of the coarse-grid solver, which is expensive for large-scale simulations, particularly at exascale and beyond. In a forthcoming article, we demonstrate that omitting post-smoothing and using pre-smoothing with 4th-kind Chebyshev polynomials of degree $2k$ is often superior to pre- and post-smoothing with degree k.

Table 10 summarizes the gains with this strategy for several CEED and NekRS test cases with spectral elements of order $N = 7$. Kershaw is a Poisson solve in the unit cube where $= 1$ implies that the mesh is a uniform array of $E = 36^3$ elements ($n = 16$M), while $\longrightarrow 0$ induces an increasing level of mesh distortion and anisotropy. The other three cases are Navier-Stokes solutions in pebble-bed geometries that are relevant to DOE's Nuclear Energy Advanced Modeling and Simulation (NEAMS) project. The number of elements for $S = 67$, 146, and 1568 pebbles are respectively $E = 122$K (n=42M), 62K (n=21M), and 524K (n=180M). The table presents the method that was found to be the fastest for each test case. (NekRS will automatically select this method.) The options include 1st-kind and 4th-kind Chebyshev smoothing, with additive Schwarz (ASM) or restricted additive Schwarz (RAS), and with polynomial smoothing of degrees (k_d k_u) on the respective downward and upward legs of the pMG V-cycle. T_S is the solve time and T_D is the solve time for the default NekRS solver. Also indicated are the gains in the coarse-grid solve times. The results show that the one-sided smoothing is not always the fastest (which is supported by our analysis), but that it does pay o for challenging mesh con gurations. Overall, the gains are anywhere from 17% to 81% over the Chebyshev-accelerated Schwarz smoothing cycle that is the current default in NekRS.

4.3 Algorithmic tradeoffs for implicit fluids solvers

Recently we considered a libCEED implementation of the discretization used by PHASTA [38], which we'll call ceed- uids. The discretization is an SUPG method for compressible Navier-Stokes in primitive (or conservative) variables and the performance test is for a snapshot of turbulent flow over a plate at Mach 0.12 with synthetically-generated turbulent inflow [32]. The freestream advective CFL is about 1, but much higher in the anisotropic boundary layer and much higher acoustic CFL given the low Mach number. The generalized α time integrator requires a nonlinear solve each time step. In PHASTA and when running ceed-fluids with linear elements and assembled sparse matrices, block Jacobi/ILU(0) preconditioning is an attractive choice, with SpMV and solving with triangular factors running at about the same speed, and large fraction of STREAM bandwidth. On GPUs, we observe that cuSPARSE sparse triangular solves run about 20× slower than SpMV. As discussed in the previous section, matrix-vector multiplication for vector-valued problems (like elasticity and fluids) on linear elements runs about twice as fast when using libCEED unassembled matrix representation, even without including the cost of assembling the sparse matrices.

As a GPU baseline, we considered GMRES(30) preconditioned by block Jacobi/ILU(0) via PETSc and cuSPARSE; this configuration achieved comparable execution time per time step as observed with PHASTA (a primarily CPU-based code running at typically strong scaled to around 80% efficiency). Switching to BCGS(ℓ) [17] and point-block Jacobi allowed us to forego assembled matrices and cut the execution time per time step by more than 2×. The initial implementation used cuSPARSE to handle an explicitly-assembled block-diagonal matrix, and after working with Junchao Zhang from the PETSc team to develop a new efficient GPU implementation, we observe a further 30% reduction in execution time. Table 11 shows a representative comparison on Perlmutter using a mesh with 5 million elements running on two nodes (8 GPUs). Note that switching from the `gmres/bjacobi/ilu` method (good on CPUs, relatively slow on GPUs) to `bcgsl/vpbjacobi` results in significantly higher iteration count, but it's more than paid off by faster execution efficiency, improving the $t_{0.8}$ strong scaling performance by roughly 3× as compared to PHASTA.

Method	PCApply (s/step)	Total solve (s/step)	PCApply speedup	Total solve speedup
`bjacobi/cuSPARSE`	0.730	2.849	-	-
`vpbjacobi`	0.142	2.216	5.1	1.29

Table 11: Execution time per preconditioner application and total solve time per time step for ceed-fluids on 2 nodes of Perlmutter. The new GPU implementation for variable point-block Jacobi is 5× faster than a mathematically equivalent preconditioner using cuSPARSE.

4.4 Performance tuning for the non-tensor MAGMA backend on NVIDIA GPUs

Similar to Section 2.5, the recently integrated GEMM selector has tuning data for the V100 and the A100 GPUs. Figure 31 shows the selector's decisions for the A100 GPU, which shows the importance of the MAGMA GEMM kernels. Figures 32 and 33 also show the corresponding results for the 3D diffusion benchmark. We also observe performance gains on both GPUs, especially for low order problem. The original backend seems to have focused only on larger order problem, which explains the similar performance for such problems before and after integrating the GEMM selector.

4.5 All-hex meshing for packed beds with contacting spheres

Turbulent flow through randomly packed spherical beds is found in many industrial processes in chemical and mechanical engineering. The flow of coolant through packed beds is of particular interest in the design of pebble-bed reactors, and researchers have expressed significant interest in detailed simulations that can provide insight into heat transfer in new pebble-bed designs. For simulations that resolve turbulent eddies in the flow, high-order discretizations having minimal numerical dissipation and dispersion provide high accuracy with a relatively small number of gridpoints, n. In collaboration with the ANL/NEAMS team, we have developed an all-hex meshing strategy for the interstitial space in beds of densely packed spheres that is tailored to turbulent flow simulations based on the SEM. The SEM achieves resolution through elevated polynomial order N and requires two to three orders of magnitude fewer elements than standard finite element approaches do. These reduced element counts place stringent requirements on mesh quality and conformity. Our meshing algorithm is based on a Voronoi decomposition of the sphere centers. Facets of the Voronoi cells

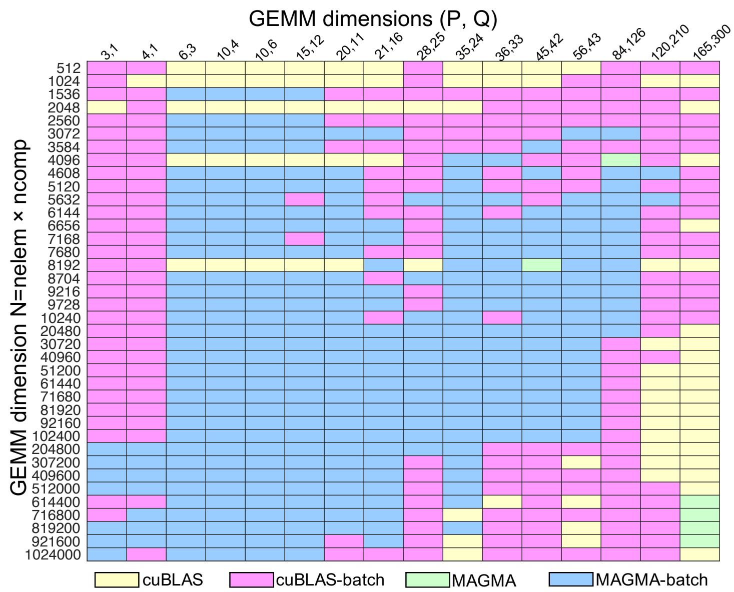


Figure 31: Colormap of the DGEMM selector for the A100

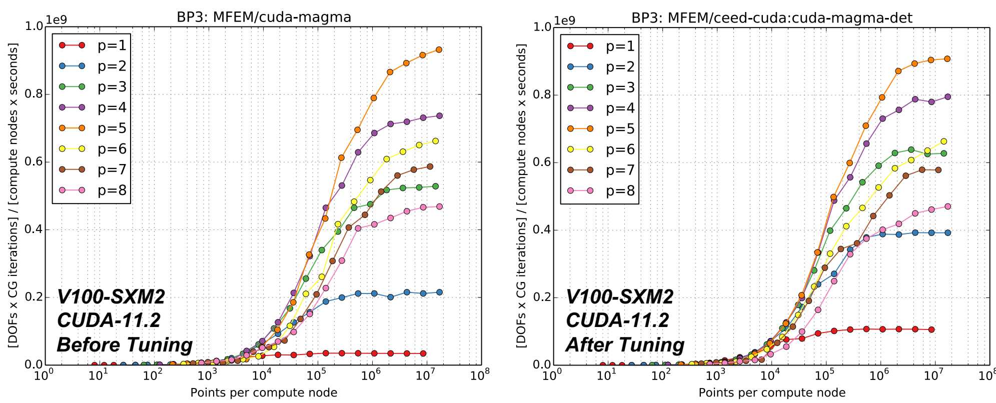


Figure 32: Performance of the 3D diffusion problem using MAGMA's non-tensor backend on the V100 GPU.

are tessellated into quad ae swept to the sphere surface to generate a high-quality base mesh. Re nements to the algorithm include edge collapse to remove slivers, node insertion to balance resolution, localized re nement in the radial direction about each sphere, and mesh optimization. A particularly important feature of our new all-hex mesh generator is the use of sphere overlap to avoid singularities at contact points. With this feature, we are able to better control the void fraction, which is important for realizing the correct overall pressure drop in the bed [1]. Using Nek5000/RS, we currently performed geometries with 146 1741 spheres using ≈300 elements per sphere (for three radial layers) for contacting spheres, along with mesh quality metrics, timings, ow simulations, and solver performance. This e ort was presented by Yu-Hsiang Lan (UIUC) [24] during his Argonne Summer Internship at the 2nd North American High Order Methods Conference (NAHOMCon) held on July 18 and 19 of 2022.

4.6 Spectral elements for coupled PNP-NS solver

In a collaborative e ort with the ANL/NEAMS team we have developed a new SEM-based Poisson-Nernst-Planck Navier-Stokes (PNP-NS) solver describing ion, uid, and thermal transport relevant to applications in the energy sector, such as battery function and nuclear reactor performance (e.g., corrosion build-up in molten

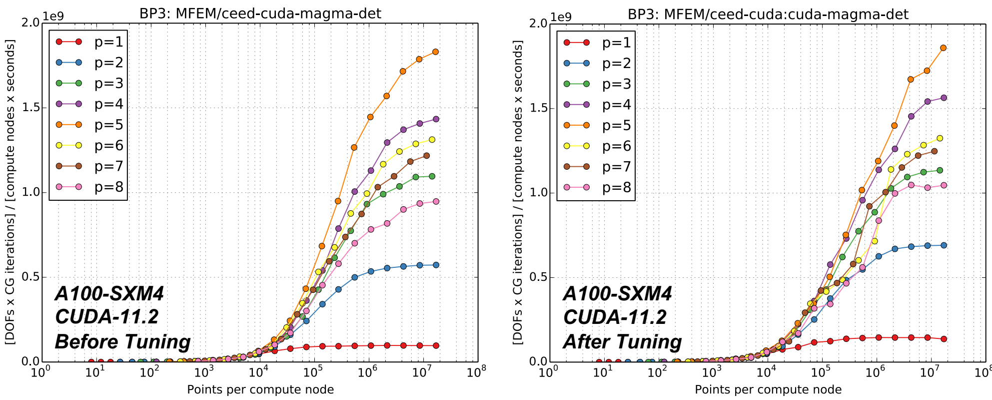


Figure 33: Performance of the 3D diffusion problem using MAGMA's non-tensor backend on the A100 GPU

salt reactors [39]). Many of the applications are in complicated domains and have high-Schmidt numbers, which makes them advection-dominated and challenging for numerical simulation. High-order methods with minimal numerical dissipation and dispersion o er the potential for reduced computational overhead for this class of problems. Our approach is to leverage the e cient and scalable parallel algorithms in Nek5000/CEM for the development of this new solver, ultimate target to be ported into NekRS for GPU runs on various architectures. The governing equations are cast in a variational framework, based on a continuous Galerkin formulation in space and a kth-order semi-implicit formulation in time using backward-di erences combined with extrapolation. We discuss important features of this complex multiphysics system, including imposition of the electro-neutrality constraint and treatment of the nonlinear electrokinetic boundary conditions. We conducted validation studies demonstrating the expected spatial and temporal accuracy for this SEM-based solver and compared with several numerical and experimental results to con rm the correctness of the proposed model and discretization. This e ort was presented by Yimin Lin (Rice University) [25] as an outcome of continuing work from his Argonne Summer Internship at the 2nd North American High Order Methods Conference (NAHOMCon) held on July 18 and 19 of 2022.

4.7 Exascale simulation workflow

A major recent development in Nek5000 and NekRS has been to extend and test the entire work ow chain to support up to 2 billion spectral elements. For typical polynomial orders, this level of support will lead to the solution of problem sizes $n = 0\ 7\ \ 1.0$ trillion grid points (3 4 trillion degrees-of-freedom), which is the anticipated problem size for full exascale runs.

When using a private-memory model such as MPI, scaling to extremely large problems does not present signi cant di culties for the bulk of the code because the *local* problem sizes are small enough that all indexing can be done with 32-bit integers. (Global indexing is e ectively based on a pair of 32-bit integers the target MPI rank and the index on that rank.) Moreover, the scalability of our solvers, both from an algorithmic and implementation standpoint, is well understood out to millions of ranks.

At setup, however, the simulation problem appears monolithic and homogeneous, so there are certain arrays where 64-bit integer indexing is required. Most of these routines have now been updated and tested in Nek5000/RS, at least out to global element counts of up to $E = 2^{31}$. Some additional routines in I/O and parallel partitioning have been updated so that they will not be bottlenecks as we move to support the largest simulations that will be enabled by exascale computing.

5. OTHER PROJECT ACTIVITIES

5.1 Sixth CEED annual meeting

The CEED project held its sixth annual meeting August 9-11, 2022 in a hybrid format: in-person at the Siebel Center for Computer Science on the UIUC campus in Urbana and virtually using ECP Zoom for videoconferencing and Slack for side discussions with participation from 88 researchers from 7 national labs, 18 universities and 6 companies. 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/ceed6am` for additional information.

5.2 Nek user meeting

A Nek5000/RS user/developer meeting was held 8/12–8/13/22. This is the first meeting since 2018. A total of 35 researchers from Europe, Asia, and the U.S. met *in person* at the meeting venue at the University of Illinois' Seibel Center for Computer Science. The agenda included a two-hour NekRS overview as well as numerous talks by senior researchers and students from applied mathematics, computer science, mechanical engineering, nuclear engineering, and aerospace. Researchers discussed issues relating to model development, performance, low-dimensional modeling, and physics. One of the most salient remarks of the meeting, made by Elia Merzari (PSU), was *Once students switch from Nek5000 to NekRS, they never want to go back!.* This remark reflects the tremendous performance gains when running NekRS on advanced GPUs, such as the NVIDIA V100 or newer processors, which is 150–200× faster than running on a single core of a CPU.

5.3 SIAM-CSE 2023 and ICOSAHOM 2023 participation

The CEED team is organizing two minisymposiums at the SIAM Conference on Computational Science and Engineering (CSE23), inviting 16 speakers from various institutions in US and Europe. Two of the CEED members are also invited to the local organizing committee of the 2023 International Conference on Spectral and High Order Methods

5.4 ALCF GPU hackathon on Polaris

Four members of Nek5000/RS team (M. Min, Y. Lan, P. Fischer, T. Rathnayake), with Kris Rowe (ALCF) and Peng Wang (NVIDIA) as mentors, participated in the ALCF GPU Hackathon on Polaris, which was held on 7/19/22, and 7/26–7/28/22. During the Hackathon activities, the team ported NekRS to Polaris and performed scaling tests for various problem sizes of the ExaSMR 17×17 rod-bundle geometry. In comparison to NekRS results on Crusher, Perlmutter and Summit, we initially found slower performance on Polaris, which initially did not support GPUDirect. During the hackathon, Polaris was upgraded to support GPUDirect. NekRS performance tests significantly improved to be slightly faster than Perlmutter (see Section 2.1 for details). During this time, our detailed performance studies identified a performance regression as Perlmutter migrated from Slingshot 10 to Slingshot 11. This regression is still being analyzed. During the hackathon, the Nek team with assistance from Aleks Obabko (ANL) also ported NRC's PANDA problem to GPUs for the first time. They provided the scientists studying PANDA with a performance report demonstrating that NekRS on Polaris provides a 10× gain over Nek5000 on Theta at the strong-scale (80% efficiency) limit.

5.5 MFEM tutorial on AWS

The MFEM team held a 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. 178 participants followed the web-based lessons in their own Amazon EC2 instances. See the tutorial page at `https://mfem.org/tutorial` for additional information.

5.6 NekRS contribution to Cardinal, 2022 R&D100 finalist

Cardinal, an open-source simulation software package for high-fidelity multiphysics solutions, was named a finalist for the 2022 R&D 100 awards. The full team of Cardinal developers includes two CEED embers (Misun Min and Paul Fischer) with Argonne Nuclear Engineers (April Novak, Richard Martineau, Elia Merzari, Paul Romano, Dillon Shaver and Patrick Shriwise). One of the key solvers integrated in the Cardinal software is NekRS. NekRS leverages high-order discretizations on high-performance graphics processing units (GPUs) to provide Cardinal users with reduced simulation times, particularly on pre-exascale and exascale leadership computing platforms that are being deployed at Argonne, Livermore and Oak Ridge National Laboratories. For more details, see `https://www.anl.gov/mcs/article/cardinal-simulation-software-named-finalist-for-2022-rd-100-awards`.

5.7 2023 INCITE submission

In collaboration with ExaSMR and NEAMS team, Nek team participated 2023 INCITE proposal submission on Full-Core RANS-LES at Exascale: Breaching the Billion Spectral Element Mark, authored by Elia Merzari, Paul Fischer, Misun Min, April Novak, and Jun Fang.

6. CONCLUSION

The goal of this milestone was to port the CEED software stack, including Nek, MFEM and libCEED to Frontier and/or Aurora early access hardware, work on optimizing the performance on AMD and Intel GPUs, and demonstrate impact in CEED-enabled ECP applications.

As part of this milestone, we also organized the next CEED annual meeting (CEED6AM) which included representatives from ECP applications, vendors and software technology projects, hosted the Nek user meeting and MFEM AWS tutorial, and worked on a number of additional software and algorithmic improvements.

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-840638.

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] Pressure drop correlation improvement for the near-wall region of pebble-bed reactors. *Nuclear Technology*, pages 1 15, 2022.

[2] A. Abdelfattah, V. Barra, N. Beams, R. Bleile, J. Brown, J.-S. Camier, R. Carson, N. Chalmers, V. Dobrev, Y. Dudouit, P. Fischer, A. Karakus, St. Kerkemeier, T. Kolev, Y.-H. Lan, E. Merzari, M. Min, M. Phillips, T. Rathnayake, R. Rieben, T. Stitt, A. Tomboulides, S. Tomov, V. Tomov, A. Vargas, T. Warburton, and K. Weiss. GPU algorithms for e cient exascale discretizations. *Parallel Comput.*, 108, 2021.

[3] Allison H Baker, Tz V Kolev, and Ulrike Meier Yang. Improving algebraic multigrid interpolation operators for linear elasticity problems. *Numerical Linear Algebra with Applications*, 17(2-3):495 517, 2010.

[4] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Vaclav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc/TAO users manual. Technical Report ANL-21/39 - Revision 3.17, Argonne National Laboratory, 2022.

[5] Jed Brown, Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jean-Sylvain Camier, Veselin Dobrev, Yohann Dudouit, Leila Gha ari, Tzanio Kolev, David Medina, et al. libceed: Fast algebra for high-order element-based discretizations. *Journal of Open Source Software*, 6(63):2945, 2021.

[6] Jed Brown, Rezgar Shakeri, Karen Stengel, and Jeremy L. Thompson. Ratel: Extensible, performance-portable solid mechanics, 2022.

[7] Jean-Sylvain Camier, Veselin Dobrev, Patrick Knupp, Tzanio Kolev, Ketan Mittal, Robert Rieben, and Vladimir Tomov. Accelerating high-order mesh optimization using nite element partial assembly on gpus. *arXiv preprint arXiv:2205.12721*, 2022.

[8] N. Chalmers, A. Karakus, A. P. Austin, K. Swirydowicz, and T. Warburton. libParanumal: a performance portable high-order nite element library, 2020. Release 0.4.0.

[9] Noel Chalmers and Tim Warburton. Portable high-order nite element kernels I: streaming operations. *CoRR*, abs/2009.10917, 2020.

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

[11] 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.

[12] 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.

[13] Veselin A. Dobrev, Patrick Knupp, Tzanio V. Kolev, Ketan Mittal, and Vladimir Z. Tomov. HR-adaptivity for nonconforming high-order meshes with the Target-Matrix Optimization Paradigm. *Eng. Comput.*, 2021.

[14] Robert D Falgout, Ruipeng Li, Bjorn Sjogreen, Lu Wang, and Ulrike Meier Yang. Porting hypre to heterogeneous computer architectures: Strategies and experiences. *Parallel Computing*, 108:102840, 2021.

[15] N. Fehn, W. Wall, and M. Kronbichler. E ciency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible ows. 2018.

[16] Paul Fischer, Stefan Kerkemeier, Misun Min, Yu-Hsiang Lan, Malachi Phillips, Thilina Rathnayake, Elia Merzari, Ananias Tomboulides, Ali Karakus, Noel Chalmers, and Tim Warburton. NekRS, a GPU-accelerated spectral element Navier-Stokes solver. *arXiv preprint arXiv:2104.05829*, 2021.

[17] Diederik R Fokkema. *Enhanced implementation of BiCGstab (l) for solving linear systems of equations*. Citeseer, 1996.

[18] Anna Fortenberry and Stanimire Tomov. Extending MAGMA Portability with OneAPI. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Ninth Workshop on Accelerator Programming Using Directives (WACCPD 2022), to appear*, SC '22. Association for Computing Machinery, 2022.

[19] Christophe Geuzaine and Jean-Francois Remacle. Gmsh: A 3-d nite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309 1331, 2009.

[20] George Karypis and Vipin Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48:71 85, 1998.

[21] David S Kershaw. Di erencing of the di usion equation in lagrangian hydrodynamic codes. *Journal of Computational Physics*, 39(2):375 395, 1981.

[22] Tzanio Kolev, Paul Fischer, Ahmad Abdelfattah, Natalie Beams, Jed Brown, Jean-Sylvain Camier, Robert Carson, Noel Chalmers, Veselin Dobrev, Yohann Dudouit, Leila Gha ari, 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.

[23] Tzanio V. Kolev, Paul Fischer, Misun Min, Jack Dongarra, Jed Brown, Veselin Dobrev, Timothy Warburton, Stanimire Tomov, Mark Shephard, Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jean-Sylvain Camier, Noel Chalmers, Yohann Dudouit, Ali Karakus, Ian Karlin, Stefan Kerkemeier, Yu-Hsiang Lan, David Medina, Elia Merzari, Aleksandr Obabko, Will Pazner, Thilina Rathnayake, Cameron Smith, Lukas Spies, Kasia Swirydowicz, Jeremy Thompson, Ananias Tomboulides, and Vladimir Z. Tomov. E cient exascale discretizations: High-order nite element methods. *Int. J. High Perform. Comput. Appl.*, 2021.

[24] Yu-Hsiang Lan, Paul Fischer, and Misun Min. Spectral element meshing for packed beds with contacting spheres. to be submitted, 2022.

[25] Yimin Lin, Haomin Yuan, Paul Fischerb, and Misun Min. Spectral elements for coupled pnp-ns equations. to be submitted, 2022.

[26] James Lottes. Optimal polynomial smoothers for multigrid v-cycles. *preprint arXiv:2202.08830*, 2022.

[27] Y. Maday, A.T. Patera, and E.M. R nquist. An operator-integration-factor splitting method for time-dependent problems: Application to incompressible uid ow. *J. Sci. Comput.*, 5:263 292, 1990.

[28] Misun Min, Michael Brazell, Ananias Tomboulides, Matthew Church eld, Paul Fischer, and Michael Sprague. Towards exascale for wind enery simulations. submitted, 2022.

[29] 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*, 2022. accepted.

[30] S. Patel, P. Fischer, M. Min, and A. Tomboulides. A characteristic-based, spectral element method for moving-domain problems. *J. Sci. Comp.*, 79:564 592, 2019.

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

[32] Michael L Shur, Philippe R Spalart, Michael K Strelets, and Andrey K Travin. Synthetic turbulence generators for rans-les interfaces in zonal simulations of aerodynamic and aeroacoustic problems. *Flow, turbulence and combustion*, 93(1):63 92, 2014.

[33] K. Swirydowicz, N. Chalmers, A. Karakus, and T. Warburton. Acceleration of tensor-product operations for high-order nite element methods. *Int. J. of High Performance Comput. App.*, 33(4):735 757, 2019.

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

[35] Ananias Tomboulides and Misun Min. Nek5000/rs les simulations for atmospheric boundary layer ows. Technical report, Argonne National Laboratory, September 2022.

[36] Christian R. Trott, Damien Lebrun-Grandie, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Je Miles, David Poliako , 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.

[37] T. Warburton, M. Phillips, and S. Kerkemeier. Kernel tuning for high-order spectral and nite elements on modern gpus. to be submitted, 2022.

[38] Christian H Whiting, Kenneth E Jansen, and Saikat Dey. Hierarchical basis for stabilized nite element methods for compressible ows. *Computer methods in applied mechanics and engineering*, 192(47-48):5167 5185, 2003.

[39] Haomin Yuan, Nathaniel Hoyt, and Misun Min. Development, validation and veri cation of moscato: A cfd-based molten salt electrochemistry and structural corrosion simulator. In *The 2022 American Nuclear Society Winter Meeting and Technology Expo*, Phoenix, AZ, US, November 13 17, 2022.