ECP Milestone Report

Support ECP applications in their exascale challenge problem runs

WBS 2.2.6.06, Milestone CEED-MS40

Tzanio Kolev
Paul Fischer
Ahmad Abdelfattah
Zach Atkins
Adeleke Bankole
Natalie Beams
Jed Brown
Robert Carson
Jean-Sylvain Camier
Noel Chalmers
Veselin Dobrev
John Holmen
Kenneth Jansen
Stefan Kerkemeier
Yu-Hsiang Lan
Damon McDougall
Elia Merzari
Misun Min
Malachi Phillips
Thilina Ratnayaka
Kris Rowe
Mark S. Shephard
Cameron W. Smith
Jeremy L. Thompson
Ananias Tomboulides
Stanimire Tomov
Vladimir Tomov
Umesh Unnikrishnan
Arturo Vargas
Tim Warburton
James Wright III

March 31, 2023

ECP Milestone Report
Support ECP applications in their exascale challenge problem runs
WBS 2.2.6.06, Milestone CEED-MS40

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

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

March 31, 2023

ECP Milestone Report
Support ECP applications in their exascale challenge problem runs
WBS 2.2.6.06, Milestone CEED-MS40

Approvals

Submitted by:

Tzanio Kolev, LLNL CEED PI	Date

Approval:

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

Revision Log

Version	Creation Date	Description	Approval Date
1.0	March 30, 2023	Original	

EXECUTIVE SUMMARY

The goal of this milestone was to support ECP applications in the preparation and execution of their exascale challenge problem runs. We focused on multi-node scaling Frontier (both strong and weak scaling) and performed additional developments to help CEED-enhanced applications to achieve their planned FOMs.

As part of this milestone, we also ported/optimized the CEED software stack, including Nek, MFEM and libCEED to Aurora and El Capitan early access hardware, worked on optimizing the performance on AMD, NVIDIA, and Intel GPUs, and demonstrating impact in CEED-enabled ECP and external applications.

The specific tasks addressed in this milestone were as follows.

- CEED-T25 (ADCD04-95): Multi-node scaling on Frontier
- CEED-T26 (ADCD04-96): Porting and optimizations for Aurora
- CEED-T27 (ADCD04-97): Porting and optimizations for El Capitan
- CEED-T28 (ADCD04-98): Help ECP applications meet their FOMs

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The goal of this milestone was to support ECP applications in their exascale challenge problem runs. We will focus on multi-node scaling runs on Frontier (both strong and weak scaling) and perform additional developments to help CEED-enhanced applications to achieve their planned FOMs.

As part of this milestone, we also ported/optimized the CEED software stack, including Nek, MFEM and libCEED to Aurora and/or El Capitan early access hardware, and worked on optimizing the performance on AMD and Intel GPUs, and demonstrating impact in CEED-enabled ECP applications.

These activities are described in Section 2 and Section 3. Additional NDA material from runs on Sunspot are available in 6.

2. CEED PERFORMANCE ON FRONTIER, AURORA, AND POLARIS

2.1 Matrix Core Instructions on MI250X

In this section we describe progress in exploiting the AMD MI250X GPU matrix-cores for improving the efficiency of CEED BK (bakeoff kernels). To do so we developed implementations of the CEED BKs using the specialized matrix-fused-multiply-accumulate (mfma) instructions to perform double precision matrix-matrix multiplication operations. Normally these instructions are used to access 512 registers per thread instead of the usual 256 registers (i.e. to allow for more registers per thread) and to attain higher peak floating point performance than the regular vector (SIMD) operations. However, we actually used these instructions because the matrix-cores are able to access registers across SIMD lanes, bypassing the shared memory bus on each compute unit.

In Table 1 we show theoretical estimates for the double precision arithmetic intensity of the CEED BK (bakeoff kernels). We note that the arithmetic intensity of each of the kernels increase approximately linearly with polynomial degree (N), and do not exceed 11 for all kernels considered up to polynomial degree $N = 15$. Indeed the scalar (odd) BKs do not exceed arithmetic intensity of 7. This provides some context for the part of the roofline model these kernels operate in, namely the relatively low arithmetic intensity and memory bound regime.

	Arithmetic intensities (flops/byte)				
N	BK1	BK2	BK3	BK5	BK6
1	1.28	3.1	1.13	0.55	0.67
2	1.75	3.78	1.44	0.71	1
3	2.2	4.38	1.76	0.87	1.33
4	2.64	4.97	2.07	1.03	1.67
5	3.07	5.55	2.38	1.19	2
6	3.51	6.14	2.7	1.35	2.33
7	3.94	6.73	3.01	1.5	2.67
8	4.37	7.32	3.33	1.66	3
9	4.8	7.92	3.64	1.82	3.33
10	5.23	8.52	3.96	1.98	3.67
11	5.66	9.11	4.27	2.14	4
12	6.09	9.71	4.59	2.29	4.33
13	6.52	10.31	4.9	2.45	4.67
14	6.95	10.91	5.22	2.61	5
15	-	-	-	2.77	5.33

Table 1: Double precision arithmetic intensity (in flops/byte) of representative CEED bakeoff kernels.

To give more context we illustrate the roofline model for a single GCD of the AMD MI250X GPU in Figure 1. The top half of the roofline diagram is only accessible when using the CDNA2 `__builtin_amdgcn_mfma_f64_16x16x4f64` intrinsic that performs a $(16 \times 4) \times (4 \times 16)$ matrix-fused-multiply-add (mfma) operations and does so at

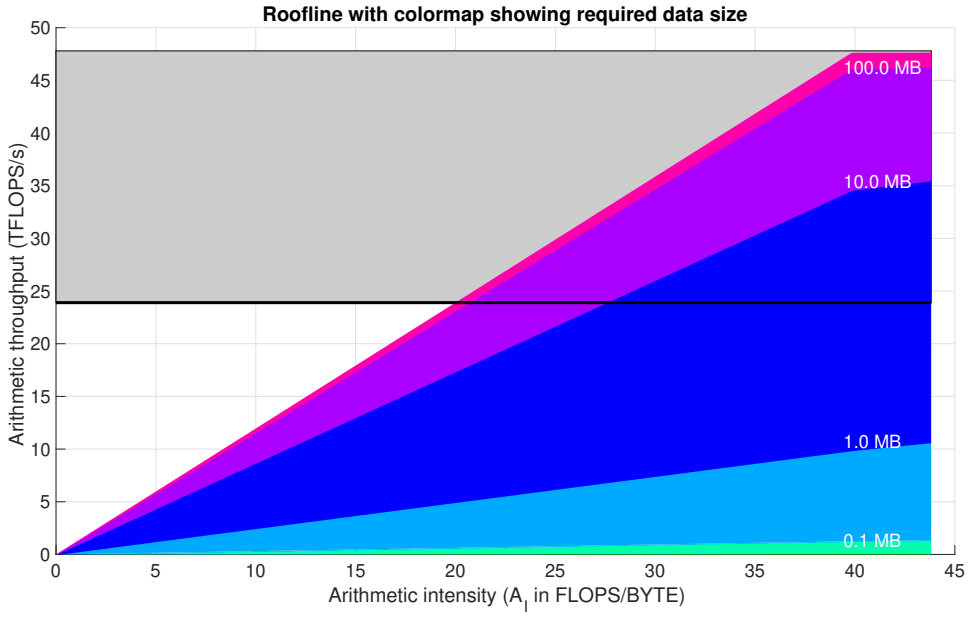

Figure 1: Roofline model for a single MI250X GCD. The quantized color scheme illustrates the data burden regimes (in megabytes) for a kernel to achieve the arithmetic throughput (vertical axis) at each arithmetic intensity (horizontal axis). This roofline was empirically calibrated using a nominal throughput of 1.2TB/s achieved by streaming kernels in the bakeoff streaming (BS) benchmark suite developed using the libParanumal library with kernels developed using the performance portability library OCCA. The upper half of the roofline diagram (illustrated by the grey box) is only accessible when using the CDNA2 double rate matrix-core instruction.

"double-rate" with theoretical peak throughput of approximately 48 TFLOPS per GCD. The output of this operation is a (16×16) matrix, requiring each of the 64 threads in the wavefront executing the instruction to dedicate four FP64 registers variables to the result. In practice we did not find it expedient to use this instruction.

The second matrix-core intrinsic instruction `__builtin_amdgcn_mfma_f64_4x4x4f64` performs four $(4 \times 4) \times (4 \times 4)$ mfma operations and does so at "single-rate" with theoretical peak throughput of approximately 24 TFLOPS per GCD. We have found this to be extremely effective when tuning the CEED BK kernels at high-order. Although it has the same peak throughput as the vector instructions it incurs significantly less shared memory bus traffic when performing the matrix-core instruction than when doing the same operation using a vector variant that uses shared memory arrays.

	HBM bandwidth (GB/s)								
	BK1		BK2		BK3		BK5 (AMD)		BK6
N	FP64	FP32	FP64	FP32	FP64	FP32	FP64	FP32	FP64
1	1286	870	1088	865	982	996	1127	1153	945
2	1224	1087	1113	976	951	937	983	1024	857
3	1177	949	979	765	1013	867	1082	1124	920
4	1134	1016	889	700	1026	901	1006	1027	781
5	1036	980	777	679	988	847	999	998	794
6	1023	1001	777	691	1037	992	1020	966	728
7	987	1022	597	545	958	961	1061	1091	913
8	897	904	567	506	974	924	1017	942	557
9	1010	892	533	470	1040	908	1006	953	676
10	830	798	707	370	1033	864	1080	945	732
11	778	766	630	372	1000	830	1113	973	845
12	843	748	690	416	965	791	1125	985	582
13	883	673	775	459	1086	913	1071	999	684
14	937	669	772	491	1018	948	1131	1065	653
15	-	-	-	-	-	-	1144	1091	774

Table 2: Maximum achieved throughputs for BK kernels in single-precision (FP32) and double-precision (FP64). When a kernel that only uses vector instructions achieved the highest throughput the result is shown in black. If a kernel that uses matrix-core instructions achieved the highest throughput the result is shown in red.

In Table 2 we show the maximum throughputs for BK1,2,3,5,6 attained in single and double precision across incrementally optimized families of kernels. The results shown in red indicate that a kernel using matrix-core instructions achieved the highest throughput of all kernels tested. It is immediately apparent that for higher polynomial degrees in both double and single precision the best performing kernels tend to be the matrix-core based versions. This is even the case where the 4×4 batch multiplies do not neatly tile the tensor-contractions in the CEED BKs and we had to resort to padding by zero. Finally, it is also interesting that the best kernels exceed 1TB/s at high orders and are close to the critical performance limitation of the MI250X GCD, namely between 1TB/s and 1.2TB/s depending on the specific details of the kernel read and writes. The single-precision kernels with relatively low streaming throughput are actually achieving 10TFLOPS in some cases at high-order.

This work was initiated by the AMD Research team including Damon McDougall and Noel Chalmers who developed the matrix-core accelerated BK5 kernels for polynomial degree $N = 15$. The Virginia Tech CEED teem continues to work with the AMD Research team to improve the performance of all CEED BK kernels for all precisions and polynomial degree.

2.2 Performance improvement for the non-tensor MAGMA backend

The non-tensor basis actions in libCEED can be implemented using a sequence of (batch) general matrix multiplications (GEMMs). Standard math libraries often provide highly optimized GEMM kernels. However,

there could be a room for improvement for very small sizes arising from low order problems, where such math libraries may not be properly tuned. In this regard, the MAGMA team has developed customized kernels to perform the non-tensor interpolation and gradient basis actions in libCEED. The kernels are based on the three assumptions. First, only two variants of the GEMM operation are assumed, either $C = A \times B$ or $C = A^T \times B$. Second, the matrix A is small enough to be cached either in the shared memory or the register file of the GPU. Third, the B and C matrices are wide enough to be subdivided into smaller matrices of the same height and the same smaller width. Each sub-matrix is assumed to fit in the shared memory or the register file of the GPU. This can be achieved through the subdivision size (or blocking size `nb`). The value `nb` is a control parameter that can be tuned based on the width of B and C.

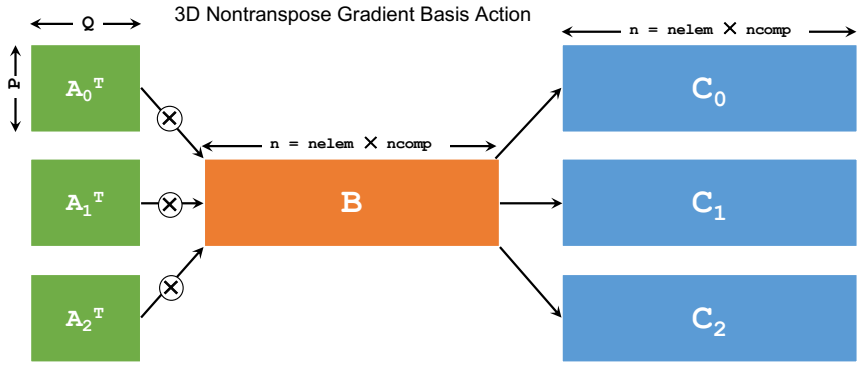

Figure 2: Non-transpose gradient basis action as a sequence of GEMMs for 3D problems

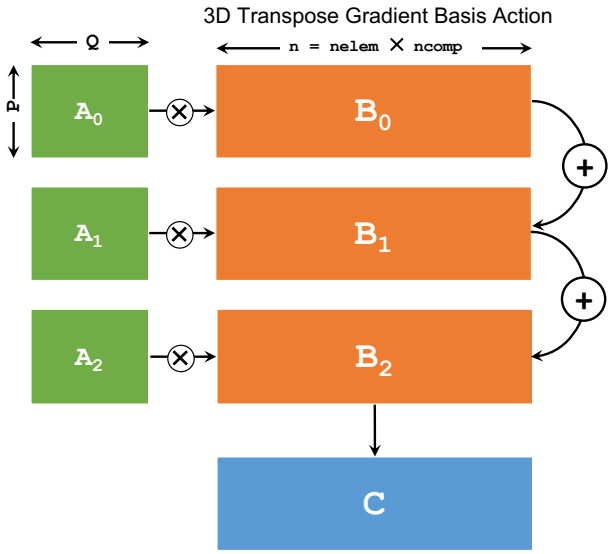

Figure 3: Transpose gradient basis action as a sequence of GEMMs for 3D problems

According to the libCEED implementation, a non-tensor interpolation basis action is performed using a single GEMM operation, while a gradient basis action consists of a sequence of GEMM calls. Figure 2 shows the sequence of calls for a non-transpose gradient action in 3D problems. All three GEMMs are independent from each other, and can all be done in parallel. However, we assume that B and C are wide enough to provide sufficient parallelism to the GPU. This is why our design uses a single thread-block (TB) to cache a sub-matrix of B for the lifetime of the kernel, and read A_0, A_1, and A_2 (one at a time) to perform the corresponding GEMM operations. A transpose gradient action is shown in Figure 3, where a reduction

operation is required across the three GEMMs to obtain the output matrix C. For this kernel, a single thread block caches a block of C for the lifetime of the kernel. Each TB performs three multiplications in order, while accumulating the results in the register file. We can observe that, in both kernels, the B and C matrices are either read or written once, which is optimal for the memory bandwidth. There are, however, redundancies in reading the A matrices, since each TB keeps its own copy locally. We believe that these redundancies have a minor effect on performance, since A is usually smaller enough to remain in the L2 cache of the GPU.

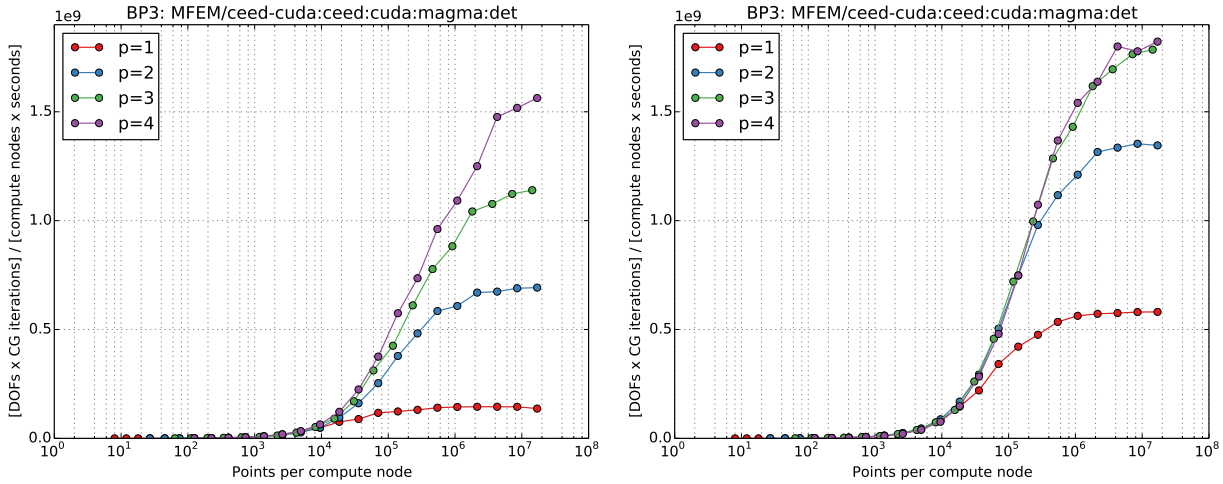

Figure 4: Performance of the MFEM 3D diffusion benchmark using the MAGMA non-tensor backend. Left is the old MAGMA backend using standard math libraries. Right is the current backend using customized kernels. Results are shown on an A100-SXM4 GPU using CUDA-11.2

The new kernels have been successfully integrated into libCEED, and have been tested using the MFEM benchmark for a 3D diffusion problem. The kernels are compiled at run time using `nvrtc` for NVIDIA GPUs, or `hiprtc` for AMD GPUs. Performance results are shown for an NVIDIA A100-SXM4 GPU (Figure 4) and for an AMD MI250x GPU (Figure 5). The performance improvements are observed only for orders up to 4. Larger order would require a more general blocking strategy. The asymptotic performance speedup on the A100 GPU are 3.85× for order 1, 1.92× for order 2, 1.53× for order 3, and 1.17× for order 4. On an AMD MI250x GPU, the asymptotic speedups are 1.97×, 1.48×, 1.3×, and 1.15×, for orders 1 through 4, respectively.

The blocking size `nb` is the only tuning parameter for the new non-tensor kernels. The MAGMA team has conducted off-line benchmark sweeps (using `nvcc` and `hipcc`) on both the A100 and the MI250x GPUs to determine the best value of `nb` for a given problem. The benchmark sweeps use the problem sizes encountered in the MFEM benchmark, and the results are tabulated in C++ structures that are looked up during the run-time compilation of the kernels.

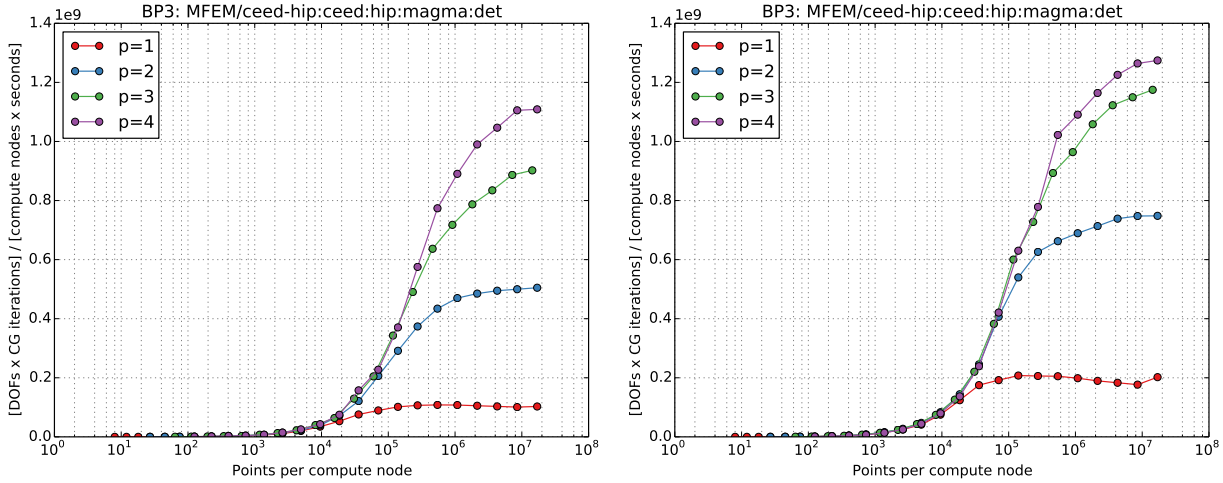

Figure 5: Performance of the MFEM 3D diffusion benchmark using the MAGMA non-tensor backend. Left is the old MAGMA backend using standard math libraries. Right is the current backend using customized kernels. Results are shown on an AMD Instinct MI250x GPU (single GCD) using ROCM-5.2

2.3 SYCL backend for libCEED

Development of a SYCL backend for libCEED began in January 2023. Led by staff at ALCF, this effort has made considerable progress and the accuracy of an initial reference implementation was verified on Sunspot and the JLSE Aurora testbed. The next phase of the project will focus on incorporating the type of kernel fusion used in the CUDA and HIP backends, as well as performance optimization. The libCEED SYCL backend will be used by the PHASTA ESP projects to meet their objectives on Aurora.

A significant engineering hurdle in the design of the libCEED SYCL backend was incorporating JIT compilation. Unlike CUDA and HIP, which have runtime compilation libraries, the current SYCL standard only supports JIT compilation indirectly. For example, using SYCL specification constants it is possible to define loop bounds at runtime before the kernel IR (e.g., SPIR-V) is lowered into a device binary. While this strategy can be used for a subset of libCEED kernels, it is not enough to support the kernel fusion model of libCEED.

Recently, Intel developed an online compiler extension for their oneAPI SYCL implementation, which allows for the runtime compilation of OpenCL source code. Using this extension, it was possible to implement kernel fusion. The downside is a small sacrifice in generality since at the present the libCEED SYCL backend is tied to a specific SYCL implementation. Therefore, to help facilitate compatibility with other SYCL implementations in the future, a hybrid approach was taken, using standard SYCL specification constants where sufficient, and the Intel online compiler extension only where necessary.

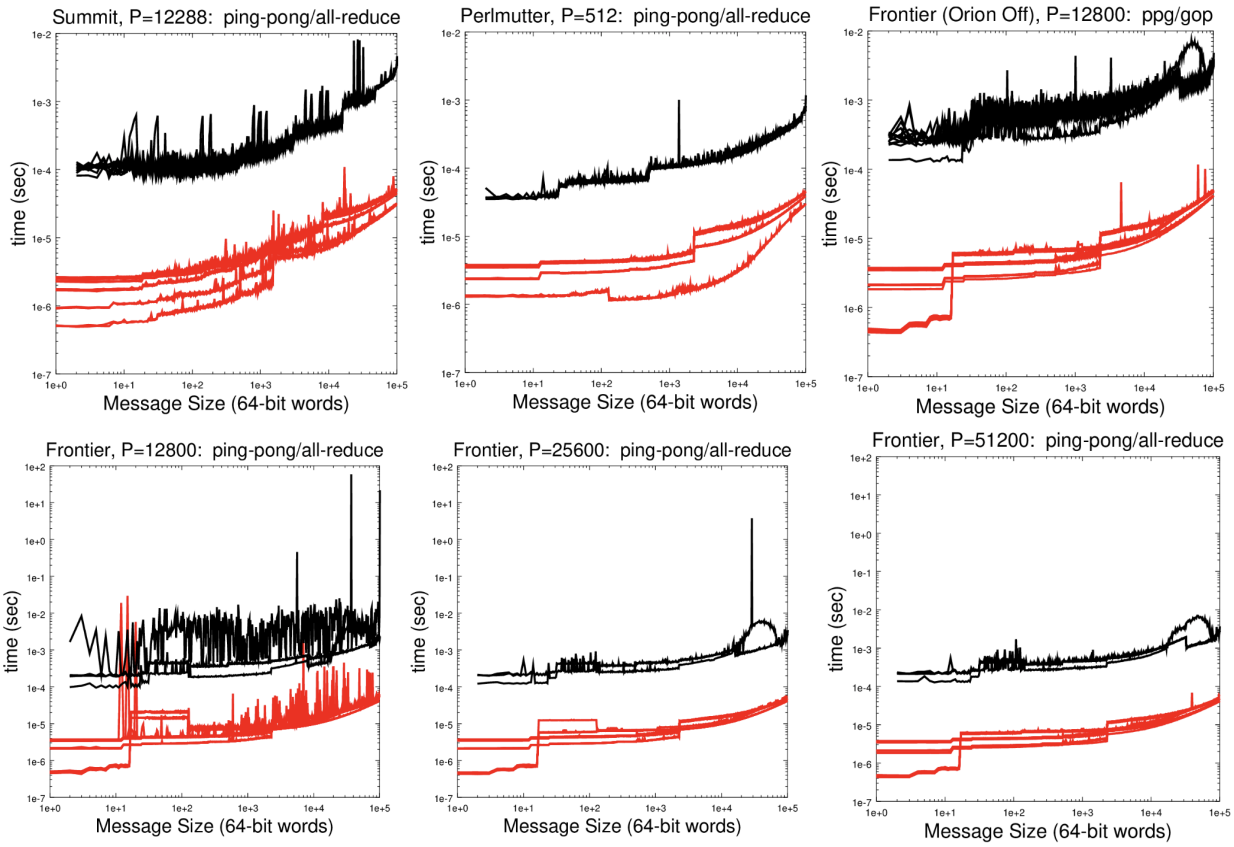

Figure 6: Ping-pong tests on Froniter (MI250Xs), Perlmutter (A100s), and Summit (V100s).

2.4 Ping-pong tests on Frontier, Perlmutter, Summit

In an effort to understand sporadic step-to-step fluctuations in NekRS wall time on Frontier, we undertook a battery of ping-pong tests using Nek5000's `platform_timer()` utility, which has been run on dozens of platforms over the past two decades. The test is a one-line change in the `.usr` file, where users prescribe case-specific functions such as initial and boundary conditions. It measures performance for tensor-contractions that are central to the spectral element method and also runs a sequence of ping-pong and all-reduce timings for message sizes ranging from 1 go 10^5 (64-bit) words. In addition to varying message sizes, the ping-pong test varies the source-destination pair, starting in all cases with rank 0 and exchanging with 64 different recipient/sender ranks, $p = 1, 2, \ldots, p_{\max} < P - 1$ (one recipient per trial). Short messages are averaged over 1000 iterations; long messages are timed for just a single round-trip. The 1/2 round-trip time is presented as the red curves in Fig. 6. The black curves represent all-reduce times over approximately 10 trials in each case.

Of particular interest in these results are the large spikes for the case of Frontier with the I/O system (Orion) turned on, seen in the lower left graph of Fig. 6. Some of the ping-pong exchanges at around 16 words averaged 0.01 seconds for the 1/2 round-trip time. Some of the all-reduce exchanges, which involve all active ranks, took 10s of seconds. This behavior was not apparent when Orion was turned off (top right plot), nor when the job occupied a larger fraction of Frontier (center and right figures in row 2). The large-job improvement presumably derives from having a single job occupy the network when those exchanges were measured. Even with Orion turned off, the short message all-reduce, relevant for vector reductions in GMRES and conjugate gradient iteration, took about 250 μsec on Frontier, vs. 100 μsec for Summit and 35 μsec for Perlmutter (using Slingshot 10). All of these results are for CPU-based messages. Data originating on the device will have different behaviors, but the CPU-based exchanges are still relevant for certain operations. We remark that no sporadic behavior was observed at lower core counts, as noted in the next Section.

Strong Scaling Test Sets			
	E	n	rank, P
Case 1	277000	95M	8–64
Case 2	470900	161M	14-128
Case 3	4709000	1.6B	128–16320

Table 3: Problem setup for strong/weak scaling studies.

2.5 NekRS scaling performance on Frontier, Crusher, Polaris, Perlmutter, Summit

We performed scaling studies for ExaSMR's 17×17 rod bundle simulations on the NVIDIA-based GPU platforms, Summit (V100), ThetaGPU (A100), Perlmutter (A100) and Polaris (A100), compared to the AMD MI-250X platforms, Frontier and Crusher [15].

In collaboration with OLCF, the Nek 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 Nek 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 2%–5% 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, and 170 layers, keeping the mesh density same, which correspond to 277 thousand spectral elements of order $N = 7$, for a total of $n = .27\text{M} \times 7^3 = 95\text{M}$ grid points, 471 thousand spectral elements of order $N = 7$, for a total of $n = .47\text{M} \times 7^3 = 161\text{M}$ grid points, and 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, respectively. Table 3 summarizes the configuration of the testing cases.

Figure 7 compares the scaling performance of Frontier to that of Crusher. Simulations are performed for 2000 steps and the average time-per-step, t_{step}, is measured in seconds for the last 1000 steps. The third-order backward-difference formula (BDF3) combined with the third-order extrapolation (EXT3) [7] is used for timestepping and the timestep size is Δt = 3.0e-04 (CFL=0.82).

Figure 7, left, shows the classic strong scaling for the problem sizes of n= 95M, 161M, and 1.6B, demonstrating the average time-per-step vs. the number of MPI ranks, P. We run a single MPI rank per GCD and there are 8 GCDs per node. The dashed lines in skyblue represent ideal strong-scale profiles for each case. The solid lines in red are for Frontier and the solid lines in black are for Crusher. We observe that Frontier is consistently slightly faster than Crusher for these three problem sizes. For larger problem sizes and processor counts, the Frontier advantage is increased.

Figure 7, right, shows the average time-per-step vs. the number of points per MPI rank, n/P, where n is the total number of grid points. t_{step} based on n/P is quite independent of the problem size, n. This is the metric illustrating that the strong-scaling performance is primarily a function of (n/P) and only weakly dependent on n or P individually, which is in accord with the extensive studies presented in [9]. Based on this metric, we can determine a reasonable number value of (n/P) for a given parallel efficiency and, from there, determine the number of MPI ranks required for a problem of size n to meet that expected efficiency. We provide more detailed performance behaviors depending on problem sizes in Figure 9.

Figure 8, left and right, shows performance for a 17×17 rod bundle with 170 layers (n=1.6B). Here we extend our discussion to other NVIDIA-based GPU architectures such as Summit (V100) at OLCF, Perlmutter (A100) at NERSC, and Polaris (A100) at ALCF, and compare those to Frontier and Crusher. While we observe that Frontier is faster than Crusher in Figure 7, we see that Crusher is faster than Summit, but not quite as fast as the A100-based Perlmutter (NERSC) and Polaris (ALCF) platforms. We provide more detailed performance behavior as a function of P in Figure 9.

Figure 9 shows the same metrics as in Figures 7–8. 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 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 $\approx$ 1 TFLOPS per rank (i.e., per A100 or GCD) for the full Navier-Stokes solution.

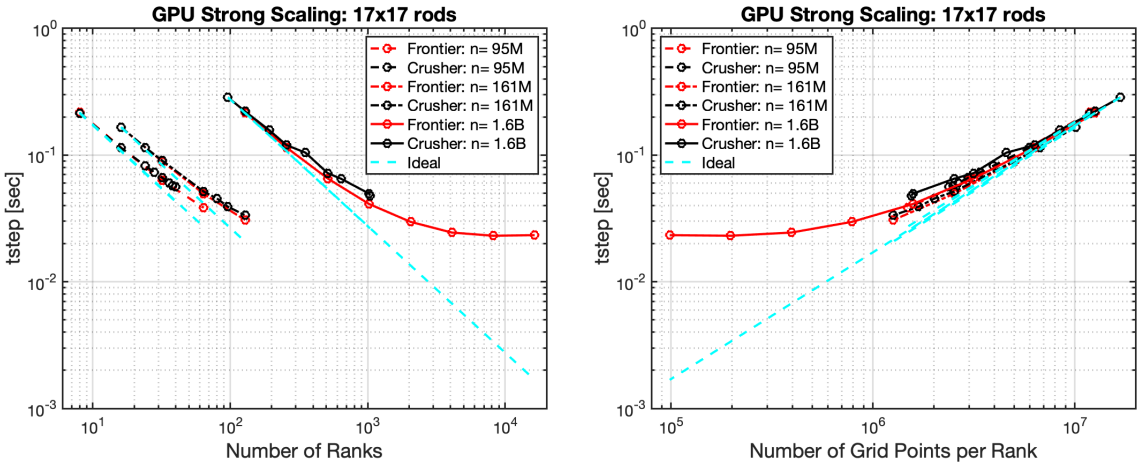

Figure 7: Strong-scaling on Frontier and Crusher for 17×17 rod bundles with 10, 17 and 170 layers with total number of grid points of $n = 95M$, $161M$, $1.6B$. Average time-per-step vs. rank, P (left) and average time-per-step vs. n/P (right). Frontier is set with (`cray-mpich/8.1.17`, `rocm/5.1.0`) and Crusher with (`cray-mpich/8.1.19`, `rocm/5.2.0`).

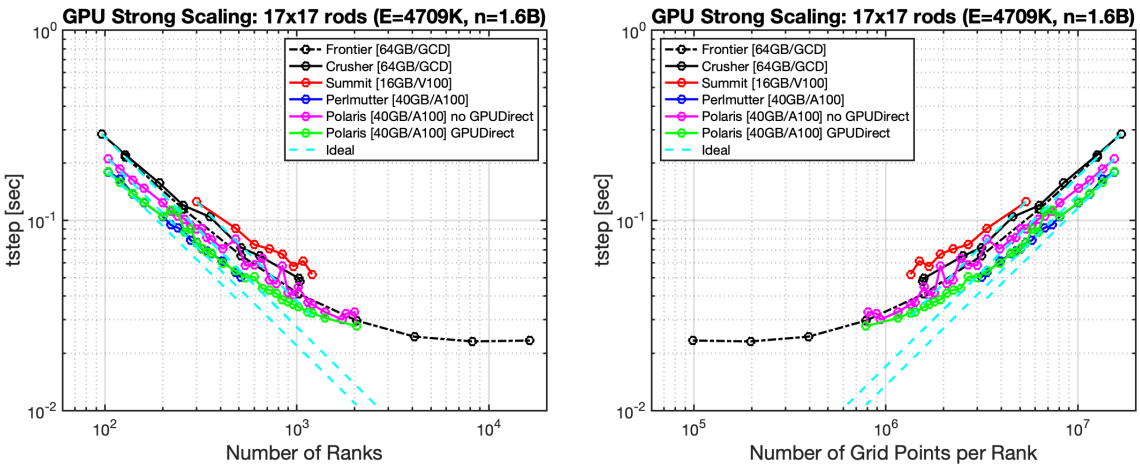

Figure 8: Strong-scaling on Frontier (MI250X), Crusher (MI250X), Perlmutter (A100), Polaris (A100) and Summit (V100) for 17×17 rod bundles with 170 layers with total number of grid points of $1.6B$. Average time-per-step vs. rank, P (left) and average time-per-step vs. n/P (right).

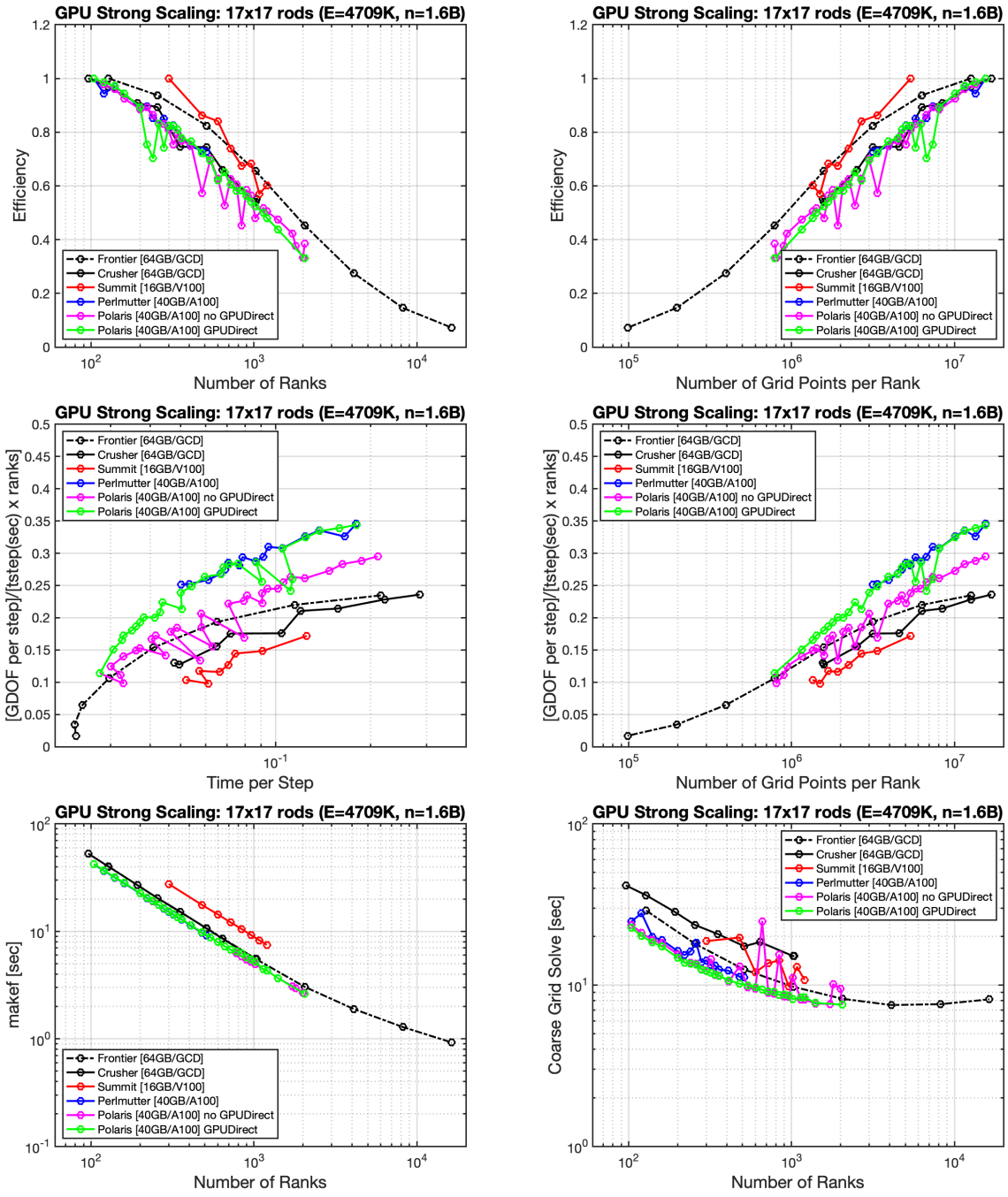

Figure 9: Strong-scaling on various GPU architectures for 17×17 rod bundle with 170 layers.

An important figure of merit is $n_{0.8}$, which is the value of n/P at which the simulation realizes 80% parallel efficiency. The plots on the first row, right of Figure 9 show $n_{0.8} = 2.5$M for Summit and 3M for Frontier. We find $n_{0.8} = 5$M for Polaris, Perlmutter, and Crusher. The plot on row 2, left, indicates that a remarkably small t_{step} value of 0.015 seconds per step is realizable on Polaris, albeit at 32% efficiency.

The plots on the last row, left, of Figure 9 show sthat the time in the advection update strong-scales quite well, as would be expected. The curves for the single GCD and A100 collapse to nearly the same performance while the older V100 technology of Summit is about 1.5× slower. In the absence of communication, this kernel is sustaining 3–4 TFLOPS FP64 on these newer architectures, although the graphs here do include the communication overhead. By 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 Summit show relatively poor performance at small values of n/P or large values of P. Also, in Figure 9 lower right we see that Polaris without GPUDirect exhibits some level of system noise.

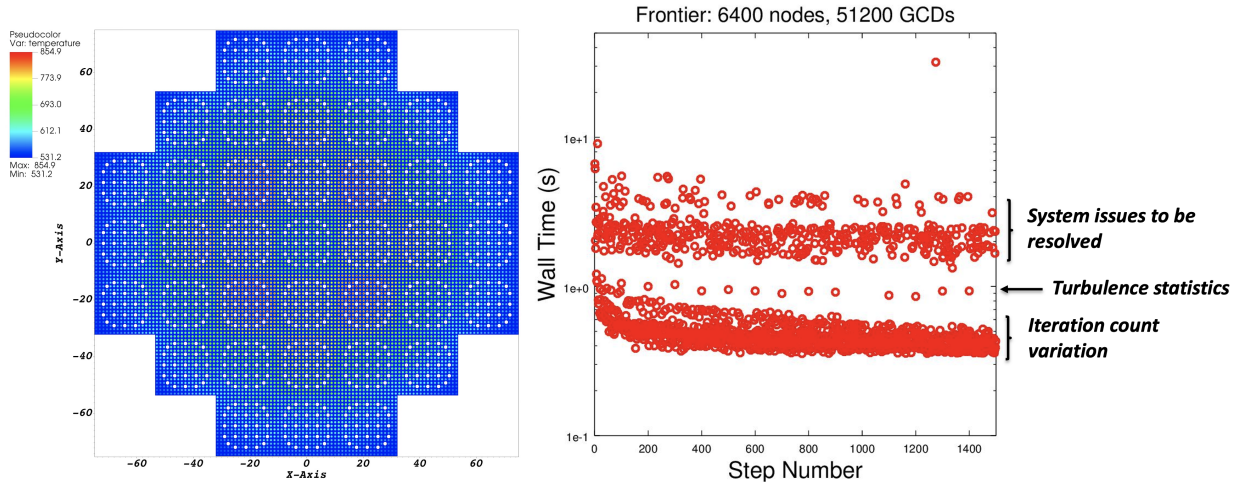

Figure 10: ExaSMR's full core 37 assemblies of 17×17 rod bundles. $E = 500M$, $N = 7$, $n = 176B$.

3. SUPPORTING CEED ECP APPLICATIONS

3.1 ExaSMR's FOM: Full-core performance on Frontier 6400 nodes

Figure 10 demonstrates NekRS performance for ExaSMR's 37 assemblies of 17×17 rod bundles, 500 spectral elements axially using 512 million elements. (Including the solid elements for this conjugate heat transfer simulation, the total element count exceeds 1 billion.) Simulations are performed for 2000 steps and the average time-per-step, t_{step} with 1.09 s/step with system noise and 0.47 s/step when neglecting system noise. The wall-clock time is broken into roughly three bands. The lowest band, which asymptotes to the interval [0.35 : 0.55] sec/step as the step number extends beyond 1200, results from standard fluctuation in NekRS iteration counts.[1] The central band, which takes roughly 1 sec per step and which occurs every 100 steps, is associated with computation of turbulent statistics on the host. Most of the statistical analyses in Nek5000 are user-developed and have not yet been ported to OCCA kernels to be GPU performant. Since they are called infrequently, porting is not a high priority at this time. The third, upper, band is currently attributed to early-access system fluctuations (e.g., as noted in Fig. 6), which are expected to improve as the system is sorted out. We reiterate that no sporadic behavior was observed at lower core counts, as noted in the preceding Section.

3.2 ExaWind: New LES modelings and convergence for atmospheric boundary layer

In collaboration with ECP ExaWind team at NREL, we have continued examining the modeling and convergence studies of two open-source codes, Nek5000/RS and AMRWind, in comparison for simulating a GABLS benchmark problem representing the atmospheric boundary layer flows [14, 16].

In this report, we summarize several models that have been developed and integrated into Nek5000/RS which are the following:

- High-pass filter (HPF) implemented through a relaxation term in the momentum equation.
- WALE (Wall-Adapting Local Eddy-viscosity) model for some simulations. (Close to the HPF results.) WMLES (Wall-Modeled Large Eddy Simulation) traction BCs at target Re (Retarget 50M): HPF fails with wall modeling at a target Reynolds number, not eddy-viscosity based.
- Splitting of eddy viscosity into an anisotropic part (mean-field eddy viscosity or MFEV) and an isotropic part which is modeled using either HPF or Smagorinsky (SMG) [21] based on fluctuation strain rate.

[1] The Nek5000/RS iterative solvers use an initial guess that is the best-fit in the space of prior solutions, which effectively removes all of the smooth error components, leaving only a random sporadic component to be solved at each step. Thus the iteration counts necessarily vary from one step to the next [10].

- SGS-TKE (subgrid-scale turbulent kinetic energy) [2] approach using filter length scale definition based on either the grid size Δ or the Deardorff scale [1].
- As an outcome, we observe the results with MFEV/SGS-TKE using a filter length scale equal to Δ agree quite well with MFEV/SMG when filter length scale is equal to Deardorff scale some deviation from MFEV/SMG.

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

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\bar{\rho}} \frac{\partial \bar{p}}{\partial x_i} - \frac{\partial \tau_{ij}}{\partial x_j} + f_i - \frac{\theta'}{\theta_0} g_i, \tag{1}$$

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

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

where an overbar denotes the LES filtering operation such that $\bar{u}_i$ is the ith component of the resolved-scale velocity vector, $\bar{\rho}$ is the density, $\bar{p}$ is the pressure, g_i is the gravity acceleration vector, and $\bar{\theta}$ is the potential temperature in the resolved scale. The scalar θ'/θ_0 that dictates the sign and strength of the buoyancy force is obtained from

$$\frac{\theta'}{\theta_0} = \frac{\bar{\theta} - \theta_0}{\theta_0}, \tag{4}$$

where θ_0 is the reference potential temperature and f_i includes the Coriolis acceleration, defined as

$$f_{c,i} = -2\epsilon_{i3k}\Omega \bar{u}_k, \tag{5}$$

where ϵ_{ijk} is the alternating unit tensor and Ω is the planetary rotation rate vector at the point of interest on the planet (which is dependent on latitude), and $j = 3$ corresponds to the vertical direction.
In addition, τ_{ij} and $\tau_{\theta j}$ are the stress tensors in the momentum and energy equations, which include (and are dominated by) SGS modeling terms

$$\tau_{ij} = -\frac{2}{Re} S_{ij} + \tau_{ij}^{sgs} = -\frac{1}{Re}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) + \tau_{ij}^{sgs}, \tag{6}$$

and

$$\tau_{\theta j} = -\frac{1}{Pe}\frac{\partial \bar{\theta}}{\partial x_j} + \tau_{\theta j}^{sgs}, \tag{7}$$

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

The MFEV-SMG is based on SMG model for isotropic part with traction boundary condition. In this model, the sub-grid-scale dissipation is again effected through a non-isotropic, MFEV obtained by the horizontally-averaged mean strain rate, and an isotropic, fluctuating part, which is here taken into account through an SMG model based on the fluctuating strain rate. In this case, the SGS model of [22] is based on the following expression

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

where also here the angle brackets $\langle\ \rangle$ denote averaging over the homogeneous directions and ν_T is an average eddy viscosity which is expressed in terms of mean flow quantities. In Eq. (8), γ is an "isotropy factor," which accounts for variability in the SGS constants due to anisotropy of the mean flow. In [22], the fluctuating eddy viscosity, ν_t, is obtained using an eddy viscosity model based on the SGS turbulent kinetic energy equation, in which the shear production term is computed from the fluctuating velocities. When the fluctuating (isotropic) part of turbulent motion is taken into account through the use of a SMG model based on the fluctuating strain rate, which is eddy-viscosity based, ν_t in Eq. (8) is non-zero and the full stress tensor has to be taken into account.

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{\partial \bar{p}}{\partial x_i} - \frac{\partial \tau_{ij}}{\partial x_j} - 2\epsilon_{i3k}\Omega \bar{u}_k + (1-\delta_{i3})\frac{\partial}{\partial z}\nu_T \frac{\partial \langle \bar{u}_i \rangle}{\partial z} - \frac{\theta'}{\theta_0} g_i \tag{9}$$

$$\approx -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\frac{1}{\mathrm{Re}} + \gamma \nu_t\right) 2S^n_{ij} - 2\epsilon_{i3k}\Omega \bar{u}_k + (1-\delta_{i3})\frac{\partial}{\partial z}\nu_T \frac{\partial \langle \bar{u}_i \rangle}{\partial z} - \frac{\theta'}{\theta_0} g_i, \tag{10}$$

$$\frac{\partial \bar{\theta}}{\partial t} + \bar{u}_j \frac{\partial \bar{\theta}}{\partial x_j} \approx \frac{\partial}{\partial x_j}\left(\frac{1}{\mathrm{Pe}} + \frac{\gamma \nu_t}{Pr_t}\right)\frac{\partial \bar{\theta}}{\partial x_j} + \frac{\partial}{\partial z}\nu_T \frac{\partial \langle \bar{\theta} \rangle}{\partial z}. \tag{11}$$

Here again, the expression for ν_T is derived so that the law-of-the-wall behavior can be recovered in the absence of any resolved turbulence. On the other hand, for the isotropic part of the eddy viscosity ν_t, the fluctuating strain rate is used:

$$\nu_t = (C_s \Delta)^2 \sqrt{2 S'_{ij} S'_{ij}}, \tag{12}$$

where

$$S' = \sqrt{2 \left\langle \left(S_{ij} - \langle S_{ij} \rangle\right)\left(S_{ij} - \langle S_{ij} \rangle\right) \right\rangle}, \tag{13}$$

and

$$C_s = \left(C_k \sqrt{\frac{C_k}{C_\epsilon}}\right)^{1/2}. \tag{14}$$

The isotropy factor γ is obtained by

$$\gamma = \frac{S'}{S' + \langle S \rangle}, \tag{15}$$

and where

$$\langle S \rangle = \sqrt{2 \langle S_{ij} \rangle \langle S_{ij} \rangle}. \tag{16}$$

Finally, the expression that holds for the horizontally averaged traction along the lower wall is:

$$\begin{aligned} \langle \tau_{uw} \rangle &= -\left(\langle \nu_t \gamma \rangle + \nu_T\right) \frac{\partial \langle u \rangle}{\partial z}, \\ \langle \tau_{vw} \rangle &= -\left(\langle \nu_t \gamma \rangle + \nu_T\right) \frac{\partial \langle v \rangle}{\partial z}. \end{aligned} \tag{17}$$

Results obtained with the MFEV/SMG approch also demonstrate convergence with increasing resolution as well as asymptotic convergence with Re and z_1^+. Moreover, convergence with resolution seems to be faster with MFEV/SMG as compared with the MFEV/HPF approach described in the previous subsection. This can be observed in the top of Figure 11, which shows horizontally averaged streamwise and spanwise velocities at t=6h, t=7h and t=8h using MFEV/SMG and traction boundary conditions at the lowest two resolutions.

Figure 11, bottom, shows the horizontally averaged streamwise and spanwise velocities at t=6h for the two highest resolutions 512^3 and 1024^3 for MFEV/HPF and for 512^3 for MFEV/SMG and AMRWind, respectively. As can be observed, both Nek5000/NekRS approaches converge to the same profiles as resolution is increased; they also agree well with the AMRWind obtained profiles at 512^3.

3.3 MAGMA in MARBL

The MARBL team has started to integrate some of MAGMA's dense matrix routines. Inversion and action of local dense matrices are common operations in all modules of MARBL. These matrices result from the L_2 FE space discretization of the material-dependent specific internal energies and group-dependent radiation energy unknowns. The initial MAGMA integration has been targeting MARBL's radiation-hydrodynamics module. This module solves a nonlinear problem through inexact Newton's method, where the resulting Jacobian computation requires inverting local L_2 dense matrices for each mesh element. These are square matrices that vary in size throughout the mesh; the number of entries for an element E is

$$\Big((\#\text{present-materials-in-E} + \#\text{radiation-groups}) \times N_E\Big)^2,$$

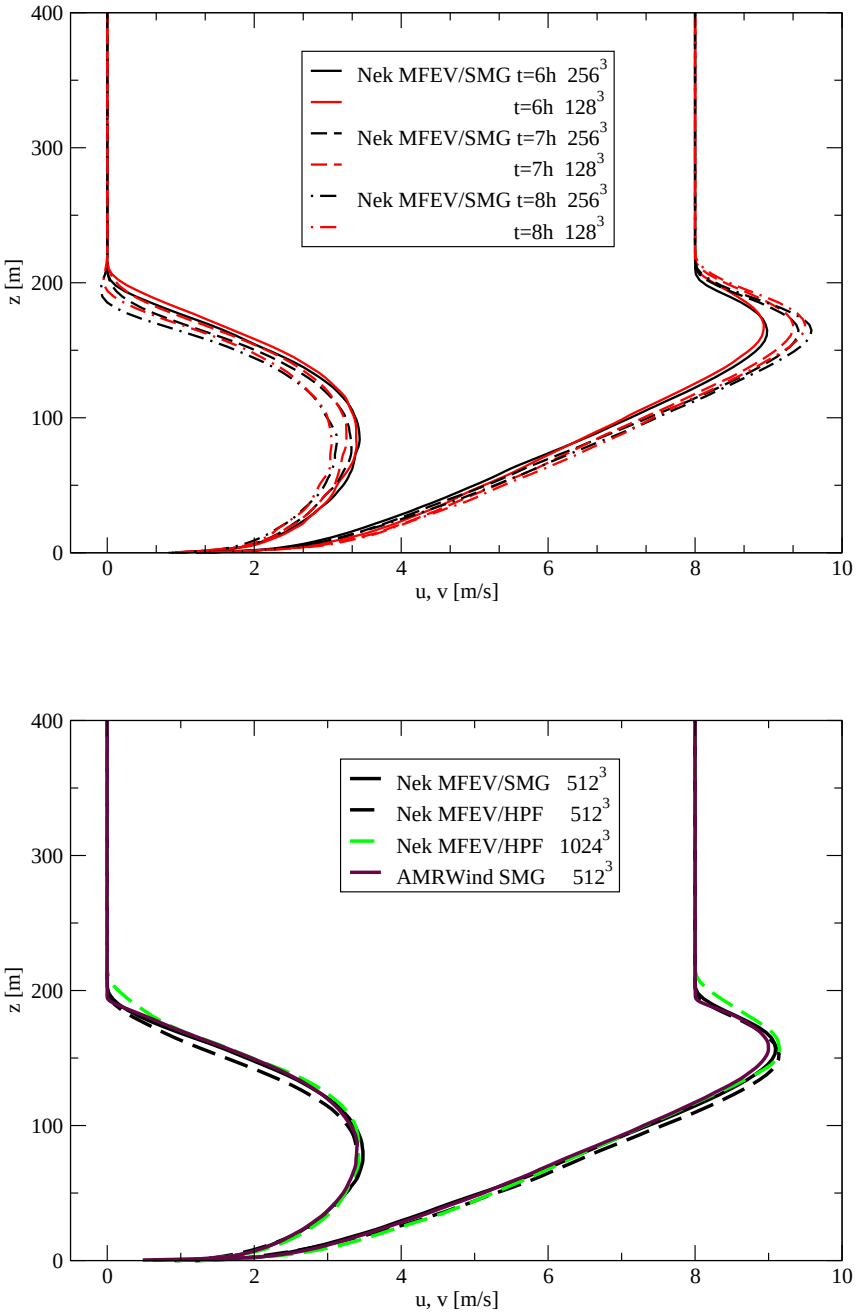

Figure 11: (Top) Horizontally averaged streamwise and spanwise velocities at t=6h, t=7h and t=8h using Nek5000 with MFEV/SMG and traction boundary conditions at two different resolutions; (Bottom) Horizontally averaged streamwise, spanwise velocities at t=6h using MFEV/SMG and MFEV/HPF with traction boundary conditions, compared with AMRWind results, for 512^3.

where N_E is the number of L_2 DOFs in E. This setup is appropriate for MAGMA's variable batch LU factorization capability, and in particular the method `magma_dgetrf_vbatched`. We report a run on 38720 3D elements on 8 V100 GPUs (4840 elements per device, Q2-Q1 discretization with $N_E = 8$). Using MARBL's baseline custom GPU code for the LU factorization, the inversion of all matrices takes 1.903s. Replacing that with MAGMA's `magma_dgetrf_vbatched` procedure, the inversion takes 0.181s, a 10.5× speedup.

More tests will be performed in the future, and more MAGMA routines will be utilized in various MARBL modules. The team is currently looking to replace existing HIP/cuBLAS calls with MAGMA ones as they have been observed to be faster than CUDA and HIP versions. These include use of the `magma_dgetrf_batched` routine for batch LU factorization, and the `magmablas_dgemv_batched` routine for matrix-vector products. These will be used in MARBL's Lagrangian and field remap phases [24], either for LU factorization of the local matrices, or for element-local action of the operators in the settings of action-based CG solves through partial assembly.

The needs of the MARBL team have prompted enhancements of batch GEMV support within MAGMA itself; these are now available in MAGMA release 2.7.1. MAGMA provides the batch matrix-vector multiplication routine (batch GEMV: $Y_i = \alpha A_i \times X_i + \beta Y_i$, for $i \in \{0, 1, \cdots,$`batch_size-1`$\}$). However, its interface previously accepted only pointer arrays that should be resident in the GPU memory before the kernel launch. A simpler interface is possible if the matrices/vectors are equidistant from each other, for which a pointer array can be replaced by a single pointer and a fixed stride. The codebase of the batch GEMV routine has been updated to support both interfaces, accepting either pointer arrays or a single pointer plus a fixed stride. Additionally, special performance optimizations have been added for small square matrices up to 32×32. A customized kernel has been developed that addresses the sub-warp configurations and maximizes the utilization of the memory bandwidth.

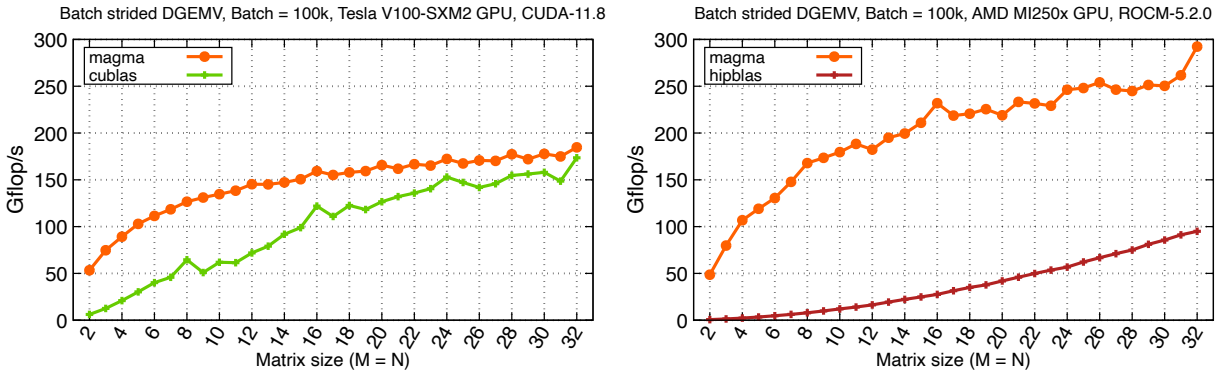

Figure 12: Performance of the batch GEMV operation in double precision. Results are shown for 100k square matrices of sizes up to 32×32. The performance tests are conducted on the two GPUs architectures powering Summit (V100) and Frontier (MI250x) supercomputers.

Figure 12 shows the performance results of the batch-strided GEMV kernel in MAGMA against the vendor libraries. The figure shows the performance gains on the V100 GPU (which powers the Summit supercomputer), and the MI250x GPU (which powers the Frontier supercomputer). The performance gains against cuBLAS on the V100 GPU range from 1.06× up to 8.65×. On the MI250x GPU, MAGMA achieves significant speedups against hipBLAS, ranging from 2.87× to 71.44×. Figure 13 shows that the MAGMA superior performance is portable to two other GPUs. On the A100 GPU, MAGMA is able to achieve speedups between 1.13× and 5.96× compared to cuBLAS. On the H100 GPU, the performance gains are between 1.08× and 5.82×.

3.4 libCEED/Fluids and PHASTA

The CU Boulder team has been collaborating with Ken Jansen to port the numerics in PHASTA to libCEED. We have focused on scale-resolving low-Mach turbulence using a stabilized (SUPG/VMS) continuous finite element formulation on unstructured meshes. The formulation solves for primitive variables, which is

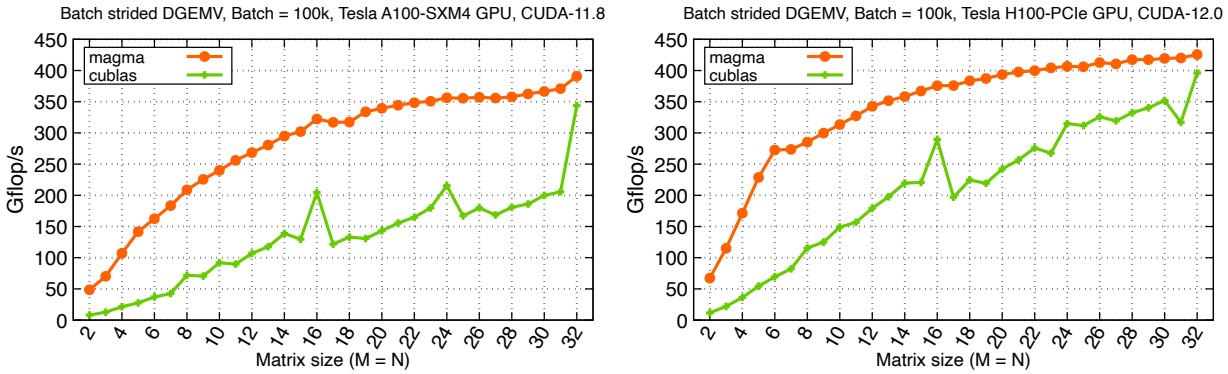

Figure 13: Performance of the batch GEMV operation in double precision. Results are shown for 100k square matrices of sizes up to 32×32. The performance tests are conducted on the A100 and the H100 GPUs to show performance portability.

applicable to all speeds while avoiding ill conditioning of the conservative basis at low Mach, and provides better accuracy in boundary layers as explained in Figure 14. The formulation is

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix}\rho\\ \rho\boldsymbol{u}\\ \rho e\end{pmatrix}}_{\boldsymbol{q}} + \nabla\cdot\left(\underbrace{\begin{matrix}\rho\boldsymbol{u}\\ \rho\boldsymbol{u}\otimes\boldsymbol{u}+pI\\ (\rho e+p)\boldsymbol{u}\end{matrix}}_{\boldsymbol{F}_{\text{inv}}}\quad\underbrace{\begin{matrix}-\boldsymbol{\sigma}\\ -\boldsymbol{\sigma}\cdot\boldsymbol{u}-k\nabla T\end{matrix}}_{\boldsymbol{F}_{\text{diff}}}\right) - \underbrace{\begin{pmatrix}0\\ \rho\boldsymbol{g}\\ \rho\boldsymbol{u}\cdot\boldsymbol{g}\end{pmatrix}}_{\boldsymbol{S}} = 0 \tag{18}$$

where the total flux $\boldsymbol{F} = \boldsymbol{F}_{\text{inv}} + \boldsymbol{F}_{\text{diff}}$ is the sum of inviscid and diffusive fluxes, $\boldsymbol{y} = [p, \boldsymbol{u}, T]$ are the primitive variables (pressure, velocity, and temperature) represented in the finite element space, and σ is the Cauchy stress. An equation of state is necessary to convert from primitive to conservative variables $\boldsymbol{q}(\boldsymbol{y})$. The time derivative is discretized as $\partial\boldsymbol{q}/\partial t = (\partial\boldsymbol{q}/\partial\boldsymbol{y})(\partial\boldsymbol{y}/\partial t)$ where $\partial\boldsymbol{y}/\partial t$ is defined in terms of $\boldsymbol{y}$ at the current step by the generalized alpha time integrator provided by PETSc. The formulation is fully implicit, typically needing 3 Newton iterations per time step.

Galerkin finite element methods are not stable for high Reynolds number flows so we follow the stabilization approach in [25], leading to the weak form

$$\begin{aligned}\int_\Omega \boldsymbol{v}\cdot\left(\frac{\partial\boldsymbol{q}}{\partial t} - \boldsymbol{S}(\boldsymbol{q})\right) - \int_\Omega \nabla\boldsymbol{v}:\boldsymbol{F}(\boldsymbol{q}) + \int_{\partial\Omega}\boldsymbol{v}\cdot\boldsymbol{F}(\boldsymbol{q})\cdot\widehat{\boldsymbol{n}}\\ + \int_\Omega \nabla\boldsymbol{v}:\left(\frac{\partial\boldsymbol{F}_{\text{inv}}}{\partial\boldsymbol{q}}\right)\boldsymbol{\tau}\left(\frac{\partial\boldsymbol{q}}{\partial t} + \nabla\cdot\boldsymbol{F}(\boldsymbol{q}) - \boldsymbol{S}(\boldsymbol{q})\right) = 0\,,\quad \forall\boldsymbol{v}\in\mathcal{V}\end{aligned} \tag{19}$$

where $\boldsymbol{\tau}\in\boldsymbol{R}^{5\times5}$ is an intrinsic time scale matrix that depends on the state $\boldsymbol{y}$ and local mesh invariants. The SUPG technique and the operator $\frac{\partial\boldsymbol{F}_{\text{inv}}}{\partial\boldsymbol{y}}$ (rather than its transpose, which appears in Galerkin Least Squares [19]) can be explained in the variational multiscale (VMS) framework via an ansatz for subgrid state fluctuations $\tilde{\boldsymbol{y}} = -\boldsymbol{\tau}\boldsymbol{r}$ where $\boldsymbol{r}$ is the strong form residual. The boundary integral appearing in (19) is not integrated literally, but requires boundary conditions.

We have verified the new solver via a range of benchmark problems and machine-epsilon agreement with PHASTA, a mature solver with two Aurora ESP projects, prior INCITE awardee, etc. The new solver produces equivalent results on CPU and GPU. We have implemented a few features that improve efficiency and numerical fidelity relative to PHASTA.

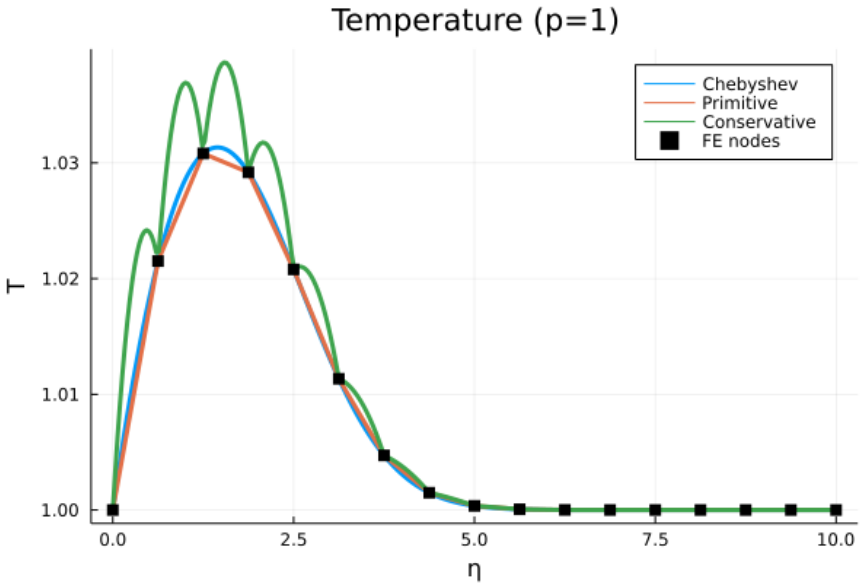

Figure 14: Nodally exact temperature solution to the compressible Blasius profile using linear interpolation in primitive versus conservative variables. Kinetic energy grows quadratically in the boundary layer, so use of a conservative basis produces "scallopping" of the temperature profile and thus inaccurate heat fluxes in the boundary layer. The Chebyshev solution is exact to 10 digits.

3.4.1 Efficient solvers

While PHASTA typically assembles sparse matrices or uses a finite difference JFNK approach with point-block Jacobi or matrix-based preconditioning, we opt for a matrix-free analytic Jacobian of the Galerkin part of the discretization (including boundary conditions) with lagging of the stabilization $\boldsymbol{\tau}$. While this lagging spoils quadratic convergence of Newton, it converges nearly as fast for the first three iterations, which provide sufficient accuracy.

While matrix-based preconditioners such as block Jacobi/ILU often pay off for linear elements on CPUs, sparse matrices are unaffordable for high order elements and triangular solves on GPUs operate at about 20x lower throughput than SpMV. Matrix-free application of the Jacobian are so much faster than matrix-based methods on GPUs that we have found point-block Jacobi adequate. LibCEED computes the numeric entries on device and PETSc's `MatSetValuesCOO` performs the necessary communication on-device. In Figure 15, we compare $t_{0.8}$, the execution time per time step for 80% efficiency strong scaled simulation using ceed-fluids on Polaris (4x A100 per node). By comparison, PHASTA running on Skylake requires about $t_{0.8} = 10$ seconds/step for linear Q_1 elements in equivalent flow regimes.

3.4.2 Boundary conditions

Our flow simulations use a synthetic turbulence generation (STG) boundary condition [20] to reduce cost to create well developed boundary layers at $Re_\theta \approx 1500$ and higher. The STG inflow is not spanwise periodic even when the simulation is (as in the NASA Speed Bump) and acts as a loud acoustic source that contaminates the solution if not damped. We use an internal damping layer (IDL) to damp these acoustics out without disrupting the synthetic structures developing into natural turbulent structures. The implementation is a ramped volumetric forcing term (similar to that described in §8.4.2.4 of [6]),

$$S(\boldsymbol{q}) = -\sigma(\boldsymbol{x}) \left.\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{y}}\right|_{\boldsymbol{y}} \boldsymbol{y}'$$

where $\boldsymbol{y}' = [p - p_{\text{ref}}, \boldsymbol{0}, 0]^T$ is a pressure-primitive anomaly and $\sigma(\boldsymbol{x})$ is a user-defined linear ramp from a max value of approximately the inverse time step reducing to zero not far from the inflow boundary.

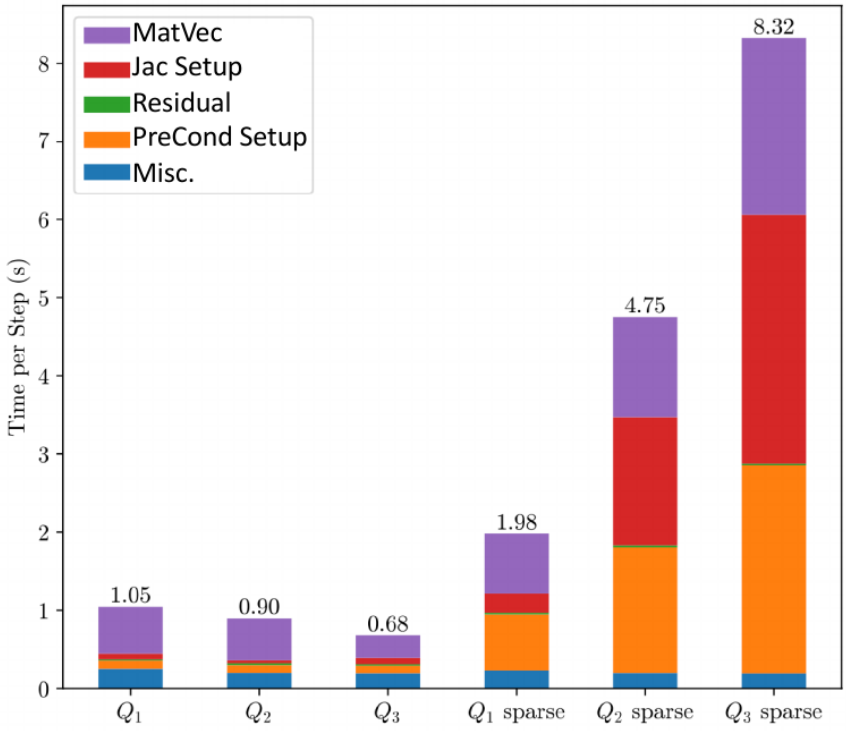

Figure 15: Cost per time step ($t_{0.8}$) of matrix-free formulations and assembled `sparse` matrix formulations.

If you know the complete exterior state, a freestream boundary condition (defined by solving a Riemann problem using a prescribed "freestream" state) is the least reflective boundary condition, but is disruptive to viscous flow structures. If thermal anomalies must exit the domain, the Riemann solver must resolve the contact wave to avoid reflections. For example, an HLLC solver is sufficient in this regard while the simpler HLL converts thermal structures exiting the domain into grid-scale reflecting acoustics (cf. [13], which was tantalizingly close to making this realization had they run one of their tests just a bit longer).

The problem is more open if acoustic reflections are not a concern and/or the flow is impacted by walls or interior structures that you wish to resolve to near the boundary. We have developed an outflow solver that combines a Riemann solve using specified exterior pressure and temperature (but extrapolated interior velocity) with a viscous flux integral that is empirically stable even in the presence of vigorous recirculation and vortices crossing lateral boundaries. We anticipate this will permit smaller domains with shorter time scales even for complex flows like the Common Research Model (CRM), in which the High-Lift Prediction Workshop recommends meshes with a spherical domain radius of 12.5 km.

3.4.3 Preparing for the NASA Speed Bump

NASA's Speed Bump represents one of the simplest problems in which RANS fails catastrophically, especially in predictiong the friction coefficient and flow separation. It has been extensively studied and quality wind tunnel data is available, thus is a good target to use DNS as a lens to explain what ingredients are missing in RANS models and would need to be added to have a chance of predicting high-lift flows "for the right reasons" (to use summary language from NASA's Fourth High-Lift Prediction Workshop). Previous DNS at $Re_L = 10^6$ [3] and subsequent studies at $Re_L = 2 \cdot 10^6$ (paper in preparation) established 6-15 nominal grid spacing (6 plus units in spanwise direction, 15 plus units in streamline direction with graded wall normal spacing starting from 0.3 plus units at the wall and reaching about 10 when entering the outer boundary layer) as DNS resolution. PHASTA's unstructured linear prism/tetrahedral elements smoothly adapted to the Kolmogorov scale enabled a nearly 5x reduction in the number of grid points required for DNS as compared to prefactored fourth order compact finite differences on overset grids [23]. Note that in the $Re_L = 1 \cdot 10^6$

Figure 16: Velocity structure for Ma = 0.1 flat plate with STG inflow, freestream top, and the new outflow boundary condition, with the boundary layer developing from $\mathrm{Re}_\theta = 970$ to 1500.

case, the flow experiences partial relaminarization while the inner boundary layer stays fully turbulent at the higher Reynolds number.

We are in the process of analyzing dispersion properties for mixed topology meshes and high order elements. Our primary test has been on the flat plate depicted in Figure 16 developing from $Re_\theta = 1000$ to 1500 using 12-30 nominal grid spacing (24-60 for quadratic and 36-90 for cubic elements), which is slightly sub-DNS resolution when using linear elements. While previous studies needed 25 to 40 days to simulate past the initial transient and compute converged statistics for $Re_L = 2 \cdot 10^6$, we anticipate that hex-dominant cubic elements will need less than 3 days on a modern GPU machine. (This will also make significantly larger simulations tractable on Aurora and Frontier.) In preparation for this study, we added interfaces to PETSc and corresponding use in ceed-fluids to compute variationally consistent projections of Reynolds/Favre averaged spanwise statistics on unstructured meshes with arbitrary partitions.

3.4.4 Data-driver subgrid stress modeling

Based on the performance results and DNS capability, both Aurora ESP projects that had been using PHASTA are in the process of transitioning to ceed-fluids. Among these is the online training and simulation using data-driven subgrid stress models based on [18], which preserves reference frame and π (unit) invariance while enabling arbitrary stress models such as neural networks. These models will be used for a range of complex flows including DDES for the CRM.

3.5 libCEED/Contact Mechanics

Ratel [4] is a solid mechanics solver based on libCEED and PETSc that was spun off from libCEED's mini-app for use in the CU Boulder PSAAP center. As part of that work, there is a need to model frictional contact of elastoplastic bonded crystalline materials with near-rigid platens. The first implementation used a Nitsche boundary condition and was found to work well with matrix-free p-multigrid. Although the literature [5] emphasizes consistency of the Nitsche formulation, it is not a consistent method with finite Poisson ratio on finite grids. The reason is similar to locking, with high-friction or no-friction surfaces producing Nische consistency forces that are significanly wrong and can even have the wrong sign. This is also a problem for plasticity since the Nitsche formulation requires evaluation of stress within a boundary integral, but plastic internal variables are only defined in the volume. We developed an alternate Nitsche-like formulation based on reaction forces that is variationally consistent on finite grids and yet preserves the attractive matrix-free p-MG applicability. It requires a composition of `CeedOperator` from volume to surface. We are working to support such composition more generally in libCEED while still providing all preconditioning ingredients.

4. OTHER PROJECT ACTIVITIES

4.1 Conferences: SIAM-CSE 2023, ICOSAHOM 2023, ParCFD

The CEED team organized two successful minisymposia at the SIAM Conference on Computational Science and Engineering (SIAM-CSE23) in Amsterdam (`https://www.siam.org/conferences/cm/conference/cse23`), including

16 speakers from various institutions in US and Europe. Two of the CEED members are invited to serve as Affinity Group Leader on High Order Methods.

Two of the CEED members are also invited to the local organizing committee of the 2023 International Conference on Spectral and High Order Methods (ICOSAHOM 2023 Seoul Korea) (`https://icosahom2023.org/`) where CEED is organizing a minisimposium on high-order algorithms, software and applications for exascale. One of the CEED members is invited as a plenary speaker.

Parallel Computational Fluid Dynamics (ParCFD) 2023 (`https://www.parcfd2023.org/`) is the 34th event in the ParCFD series since its inaugural event in 1989. ParCFD is an annual international forum devoted to the discussion of recent developments and applications of parallel computing in the field of computational fluid dynamics and related disciplines. One of the CEED members is invited as a plenary speaker to ParCFD 2023.

4.2 Software Release: MFEM v4.5.2

Version 4.5.2 of MFEM was released on March 23, 2023. Some of the new additions in this release are:

- Support for pyramids in non-conforming meshes.
- Removed the support for the Mesquite toolkit in favor of MFEM's TMOP algorithms.
- New fast normalization-based distance solver.
- Option to auto-balance compound TMOP metrics.
- Support for shared Windows builds with MSVC through CMake.

For more details, see the interactive documentation at `https://mfem.org`.

4.3 Software Release: MAGMA v2.7.1

In this release we added support for CUDA 12, as CUDA 12 depreciated some CUDA features previously used in MAGMA. A new interface for batch GEMV that accepts a pointer plus stride was added, and performance was improved for batch GEMV targeting square sizes up to 32. These batched GEMV operations were used in the ECP ExaAM project. See the MAGMA 2.7.1 release notes for further details. Additionally, as part of the ongoing work to port and tune MAGMA for Intel GPUs, the MAGMA team now has a publicly-visible branch of MAGMA with SYCL support.

4.4 Software Release Upcoming: NekRS v23.0

An upcoming release of NekRS v23.0 includes the following updates:

- Added point interpolation
- Added coupled multi-session (neknek)
- Added particle tracking capability
- Added single source udf+oudf
- Improved runtime statistics
- Improved Chebyshev smoother
- Support flexible time averaging
- Support 'on' boundary condition (aligned)
- Added extrapolation initialGuess method
- Support scaleable JIT compilation
- Added more examples
- Updated various bug fixes

5. CONCLUSION

The goal of this milestone was to support ECP applications in the preparation and execution of their exascale challenge problem runs. We focused on multi-node scaling Frontier (both strong and weak scaling) and performed additional developments to help CEED-enhanced applications to achieve their planned FOMs.

As part of this milestone, we also ported/optimized the CEED software stack, including Nek, MFEM and libCEED to Aurora and El Capitan early access hardware, worked on optimizing the performance on AMD, NVIDIA, and Intel GPUs, and demonstrating impact in CEED-enabled ECP and external applications.

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-XXXXXX.

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] Stratocumulus-capped mixed layers derived from a three-dimensional model. *Boundary-Layer Meteorology*, 18:495–527, 1980.

[2] Stratocumulus-capped mixed layers derived from a three-dimensional model. *Boundary-Layer Meteorology*, 71:247–276, 1994.

[3] Riccardo Balin and Kenneth E Jansen. Direct numerical simulation of a turbulent boundary layer over a bump with strong pressure gradients. *Journal of Fluid Mechanics*, 918:A14, 2021.

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

[5] Franz Chouly, Mathieu Fabre, Patrick Hild, Rabii Mlika, Jérôme Pousin, and Yves Renard. An overview of recent results on nitsche's method for contact problems. In *Geometrically Unfitted Finite Element Methods and Applications: Proceedings of the UCL Workshop 2016*, pages 93–141. Springer, 2017.

[6] Tim Colonius. Chapter 8 - boundary conditions for turbulence simulation. In Robert D. Moser, editor, *Numerical Methods in Turbulence Simulation*, Numerical Methods in Turbulence, pages 319–357. Academic Press, 2023.

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

[8] 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.

[9] Paul Fischer, Misun Min, Thilina Rathnayake, Som Dutta, Tzanio Kolev, Veselin Dobrev, Jean-Sylvain Camier, Martin Kronbichler, Tim Warburton, Kasia Świrydowicz, et al. Scalability of high-performance PDE solvers. *The International Journal of High Performance Computing Applications*, 34(5):562–586, 2020.

[10] Paul F Fischer. Projection techniques for iterative solution of Ax= b with successive right-hand sides. *Computer methods in applied mechanics and engineering*, 163(1-4):193–204, 1998. Publisher: Elsevier.

[11] Tzanio Kolev, Paul Fischer, Ahmad Abdelfattah, Adeleke Bankole, Natalie Beams, Michael Brazell, Jed Brown, Jean-Sylvain Camier, Noel Chalmers, Matthew Churchfield, Veselin Dobrev, Yohann Dudouit, Leila Ghaffari, 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, and James Wright III. Improve performance and capabilities of CEED-enabled ECP applications on Frontier/Aurora EA, September 30, 2022.

[12] Y. Maday, A.T. Patera, and E.M. Rønquist. An operator-integration-factor splitting method for time-dependent problems: Application to incompressible fluid flow. *J. Sci. Comput.*, 5:263–292, 1990.

[13] Gianmarco Mengaldo, Daniele De Grazia, Joaquim Peiro, Antony Farrington, Freddie Witherden, Peter Vincent, and Spencer Sherwin. A guide to the implementation of boundary conditions in compact high-order methods for compressible aerodynamics. In *AIAA Aviation 2014*, Atlanta, June 2014. AIAA.

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

[15] Misun Min, Yu-Hsiang Lan, Paul Fischer, Thilina Rathnayake, and John Holmen. ek5000/rs performance on advanced gpu architectures. Technical report, Argonne National Laboratory, September 2022.

[16] Misun Min and Ananias Tomboulides. Simulating atmospheric boundary layer turbulence with nek5000/rs. Technical report, Argonne National Laboratory, September 2022.

[17] 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.

[18] Aviral Prakash, Kenneth E. Jansen, and John A. Evans. Invariant data-driven subgrid stress modeling on anisotropic grids for large eddy simulation, 2022.

[19] Farzin Shakib, Thomas JR Hughes, and Zdeněk Johan. A new finite element formulation for computational fluid dynamics: X. the compressible Euler and Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 89(1-3):141–219, 1991.

[20] 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.

[21] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164, 1963.

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

[23] Ali Uzun and Mujeeb R Malik. High-fidelity simulation of turbulent flow past gaussian bump. *AIAA Journal*, 60(4):2130–2149, 2022.

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

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

6. NDA MATERIAL

This section contains NDA material, in particular all of the results on the Aurora early access systems (Sunspot, Arcticus, and Florentia) included in this report.

6.1 MAGMA in ExaAM

Initial testing of ExaConstit's HIP port on Crusher with a mesh size and loading conditions, similar to what the ExaAM Frontier runs would utilize, noted a 2x slow-down in performance in comparison to the same Summit runs. Through the use of Caliper annotations and help from HPE staff at the Crusher office hours, it was determined that the batch matrix-vector product used in the element assembly formulation was the root cause. Initial attempts to fix this issue made use of hipBLAS which did increase performance on Crusher, but the Crusher runs were still slower than Summit. After reaching out to the MAGMA team as part of the ExaAM-CEED engagement, they were able to provide us optimized batch GEMV calls that are easily usable within MFEM. These optimized kernels resulted in a large performance wins for ExaConstit on Crusher resulting in a 3x speed-up over the original Crusher runs and a 1.5x speed-up over the Summit runs. All of the timings of these runs are summarized in Table 4. Lastly, the MAGMA work was used as part of OLCF ticket: OLCFDEV-1325 to drive optimizations within hipBLAS for small matrices N ¡=32 and M ¡=32 and those optimizations should be landing in ROCm v5.6.0.

Platform	Code modification	Runtime (s)
Summit	develop branch	780
Crusher	HIP develop branch	1450
Summit	HIP develop branch	720
Crusher	HIP batch (GEMV) branch	960
Summit	HIP batch (GEMM) branch	830
Crusher	MAGMA batch (GEMV) branch	480

Table 4: Summary of batch matrix-vector optimizations within ExaConstit/MFEM for different platforms

6.2 Omega_h progress on Sunspot

The Omega_h GPU accelerated conforming mesh adaptation library supports execution on AMD and NVIDIA GPUs. In the last period efforts were focused on moving towards having a Kokkos-only backend that has no dependency on vendor libraries such as CUDA, HIP, (Roc)Thrust or Intel DPL, and runs on AMD, NVIDIA, and Intel GPUs. This backend passes all tests on NVIDIA GPUs and is failing 3 of 19 tests when ran on an Intel Ponte Vecchio GPU.

Critical to the increase in passing tests on the Ponte Vecchio GPU was (1) a compiler bug fix (CMPLRLLVM GSD2581 available in the 2022.12.30.002 compiler) and (2) support from ANL and Intel staff running tests using development drivers and compilers (SDK). Given that several tests went from failing to passing when using the development SDK, the Omega_h test suite was added to the set of tests ran by ALCF when a new SDK is made available to Sunspot users and included in a milestone report focusing on application readiness for the Intel-ANL Compiler Working Group.

6.3 NekRS, ExaSMR scaling performance on Sunspot, up to 32 nodes

We have been continuing weekly or bi-weekly meetings with Intel team. In collaboration with Intel team, we ported NekRS on Sunspot, and studied kernel performance and ExaSMR's singlerod and 17×17 rod-bundle simulation performance for varying sizes, using up to 384 tiles on Sunspot PVCs.

Singlerod Performance on a single PVC on Sunspot. Table 5 demonstrates NekRS performance for ExaSMR's singlerod simulation on a single PVC. Simulations are performed for 500 steps and the average time-per-step, t_{step}, is measured in seconds for the last 400 steps. For a given system, the ratio is t_{step} to

GPU Performance on a Single GPU: singlerod, $E = 7168$, $n = 2,458,624$, $N = 7$							
System	gpu	Device	API	v_i	p_i	t_{step} (sec)	Ratio
Summit	1	16GB V100 GPU	CUDA	4	1	6.78e-02	1.62
Sunspot	1	64GB PVC 1T	DPCPP	4	1	8.39e-02	2.01
Sunspot	1	128GB PVC 2T	DPCPP	4	1	5.97e-02	1.43
Spock	1	32GB MI100 GPU	HIP	4	1	7.98e-02	1.91
Crusher	1	64GB MI250X (1 GCD)	HIP	4	1	6.55e-02	1.57
ThetaGPU	1	40GB A100 GPU	CUDA	4	1	4.31e-02	1.03
Perlmutter	1	40GB A100 GPU	CUDA	4	1	4.17e-02	1.00
Polaris	1	40GB A100 GPU	CUDA	4	1	4.16e-02	1.00

Table 5: NekRS performance on various architectures using a single GPU.

that of Polaris. v_i and p_i represent the average iteration counts per step of the velocity components and pressure. Timestepping is based Nek5000's second-order characteristics method with one substep [12, 17] and the timestep size is Δt = 1.2e-3 (CFL=1.82). Pressure preconditioning is based on p-multigrid with CHEBYSHEV+ASM smoothing and hypre AMG for coarse grid solve [8]. Tolerances for pressure and velocity are 1e-4 and 1e-6, respectively.

The single-device results of Table 5 show that, for the current version of NekRS, the A100 is 1.62× faster than the V100 on Summit. The current performance for a single tile PVC is about half that of the A100, while two tiles (with MPI overhead) realize 70% of the A100 perfomance. A single GCD of the MI250X realizes about 64% of the A100 performance. It's important to note that the A100 and MI250X kernels have been extensivel tuned for the respective NVIDIA / AMD products, whereas the DPCPP code running on PVC is the straight output from OCCA kernels in the current NekRS release, with no platform-dependent tuning. The NekRS/ANL team will be working with Intel to optimize the key kernels, particularly the advection kernel, which puts significant pressure on registers because, due to dealiasing, the data is roughly $(3/2)^3\times$ larger than the usual $(N+1)^3$ data blocks associated with Nth-order hexahedral elements used in the spectral element formulation.

17×17 rod-bundle performance on 32 nodes of Sunspot. Figure 17 shows strong-scaling on Sunspot for 17×17 rod bundle with 10 layers. Of particular note is the lower left figure—`makef()` refers to construction of the right-hand side terms for the Navier-Stokes update, which includes the dealiased advection terms mentioned above. We see that Sunspot is not doing well for this term. On the other hand, for the communication-intensive coarse-grid solve, which is executed on the host, Sunspot is just as fast as Perlmutter and Polaris.

6.4 MAGMA and GEMMs for non-tensor finite elements on PVC

As part of the ongoing effort to port the MAGMA library to SYCL and early efforts to port libCEED to SYCL, we have investigated the performance of the batch GEMM operations used by the MAGMA backend for non-tensor basis functions on PVC. We consider both the new specialized kernels for lower-order elements, discussed in 2.2, and the "GEMM selector" used to pick the best configuration of standard and batch GEMMs from MAGMA or the vendor-provided library. The specialized kernels are incorporated into libCEED's runtime compilation framework for CUDA and HIP, but as libCEED SYCL support is still in progress, the kernels were instantiated for a range of values of P (total number of nodes in element), Q (total number of quadrature points), and `nb` (columns of each matrix in the batch GEMM operation) and compiled ahead-of-time. We considered "full" ahead-of-time compilation for the PVC device, rather than relying on SYCL's default JIT. These stand-alone testers consider only the required multiplication for the basis action of a particular size, rather than working inside an actual libCEED BP implementation. In all cases, each case was repeated eleven times, with the results from the first iteration removed as a warm-up and the other cases averaged. We consider one tile of one GPU on Sunspot, assuming we will want to tune the MAGMA backend to perform best in explicit scaling mode; further experiments using the whole GPU with implicit scaling can be done in the future.

The specialized kernels, standard MAGMA GEMM, and standard MAGMA batch GEMM were all ported from CUDA to SYCL with the help of Intel's `dpct` tool. In cases where MAGMA has separate tuning parameters for CUDA and HIP (as for the standard GEMM), the CUDA parameters were chosen as a

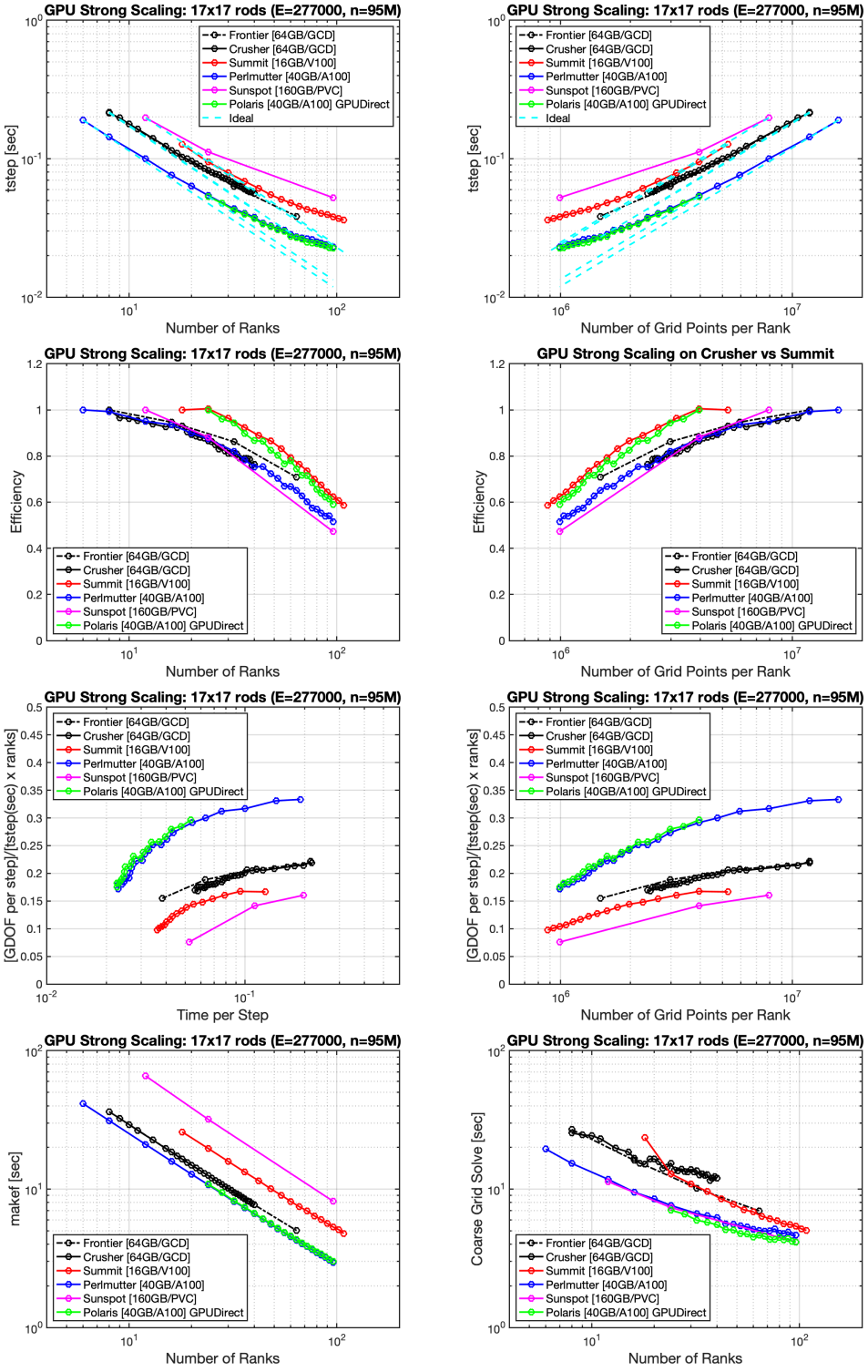

Figure 17: Strong-scaling on various GPU architectures for 17×17 rod bundle with 10 layers.

starting point. Any changes to the kernels were mostly to slightly rearrange code as necessary to allow all synchronization (`__syncthreads` in CUDA/HIP; `barrier` in SYCL) to be outside of any conditional checks. This is currently required to satisfy the stricter barrier requirements for SYCL, which does not allow threads which have exited early to be exempted from participating in the barrier before other threads can proceed.

Tuning of the specialized MAGMA backend non-tensor kernels. In the following, values of P and Q align with those used in MFEM for standard tetrahedral elements of orders 1 through 8. Values of N – the total number of elements (multiplied by the total number of components, if the basis has more than one component) – replicate the data in the lookup tables of the current libCEED MAGMA backend GEMM selector. For every N considered, `nb` was varied from 1 to 32 (including every value in between) for the specialized kernels. To find the best values of `nb`, the results of the benchmark were split into five groups based on the value of N. The groups were separated by the same process that the libCEED MAGMA backend currently uses to select a specialized kernel to run for a particular problem size. Within each group, the performance for each N was normalized by the highest-rated Gflop/s rate for that N, resulting in values ranging from 0-1 across the different values of `nb` The `nb` which maximized the minimum performance across the group was selected for the group. The top rows of Figures 18 and 19 illustrate this selection for two element sizes; the vertical lines indicate the selected value of `nb` for the group.

Large GRF compilation option. The specialized MAGMA basis kernels make heavy use of registers, and the compiler warned of large amounts of spilling when performing the ahead-of-time device code compilation for PVC. We also ran the benchmarks with the `-ze-intel-enable-auto-large-GRF-mode` compiler flag enabled. This doubles the allocated registers for the kernel and changes trends in performance across `nb` and the best `nb` to choose, as seen in the bottom rows of Figures 18 and 19. As enabling "large GRF mode" improved performance of the specialized kernels – from modest gains around 1.2X to over 3X or 4X speedup comparing the best-case "large GRF" to the best-case "regular" – and did not negatively impact standard GEMM, we consider exclusively large GRF results from this point forward.

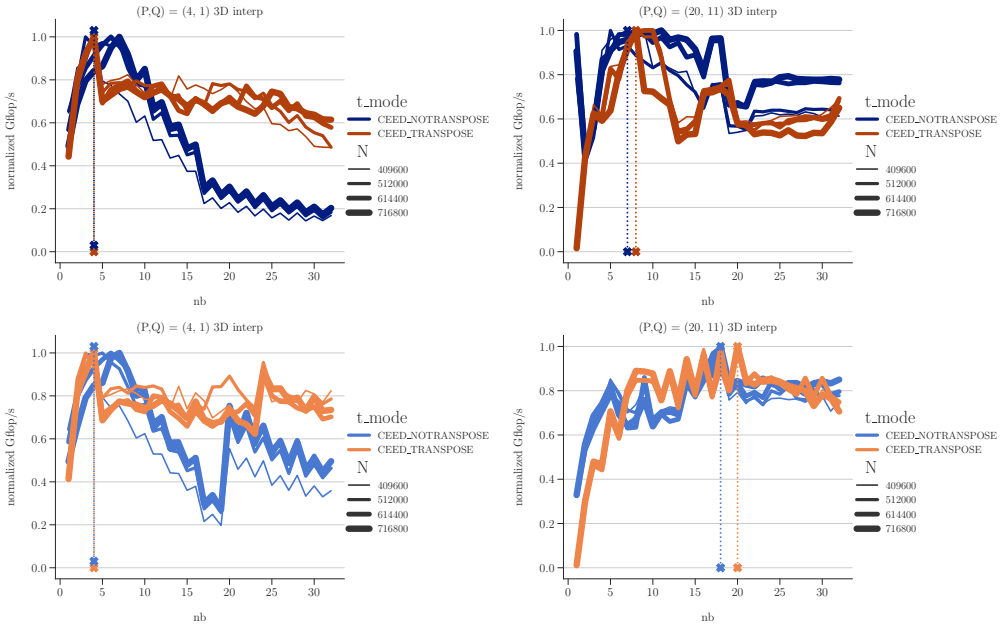

Figure 18: Selecting the best interpolation `nb` for a group of values of N, on one tile of PVC. The top row shows the default compilation flags, while the bottom row shows the version with large GRF enabled. Dashed vertical lines indicate the selected best value for `nb`.

Standard GEMMs for higher-order non-tensor elements. For higher orders of basis functions, the MAGMA backend uses standard GEMM and batch GEMM computations from either the MAGMA library

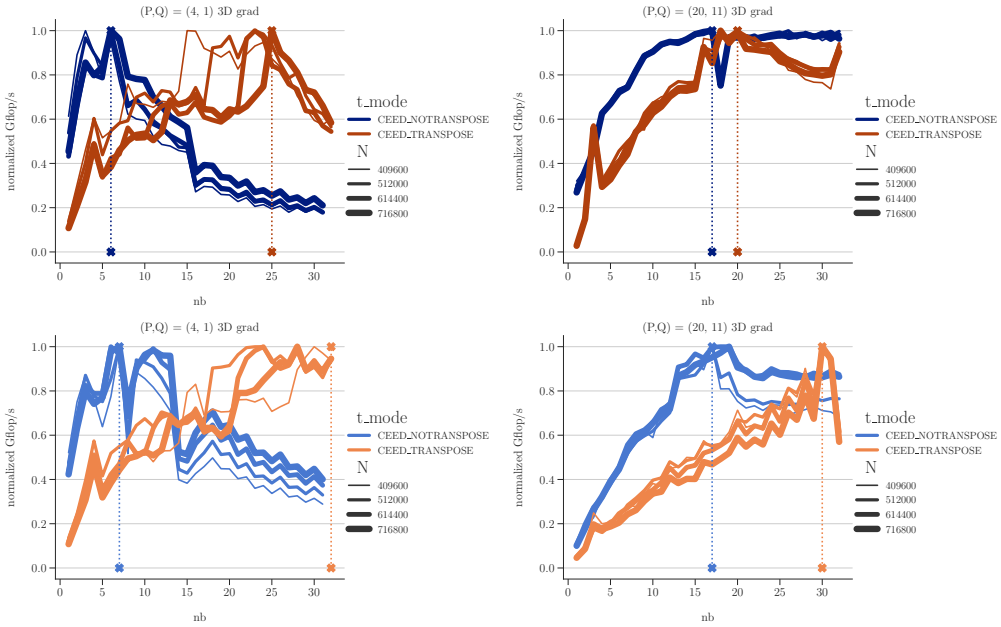

Figure 19: Selecting the best gradient `nb` for a group of values of N, on one tile of PVC. The top row shows the default compilation flags, while the bottom row shows the version with large GRF enabled. Dashed vertical lines indicate the selected best value for `nb`.

or the vendor-provided BLAS. To find the best configurations with the ported SYCL code and with oneMKL, we considered larger values of `nb`, starting at 32 and increasing in multiples of two for values that evenly divide the current N; this was again based on the previous benchmarks used to create the current GEMM lookup tables for CUDA and HIP. In Figure 20, we show an overview of the selected configurations for the transpose interpolation action. (The middle table, for MI250X, reproduces the last four columns of Figure 13 from [11], with some added information.) The color-coding of the boxes indicate the choice of routine, while the text shows the `nb` chosen if the box represents a batched GEMM routine. Note that there are four boxes marked with "*" in the right-most column of the A100 table; these are the only cases for which the GEMM choice used in the following results are different from what would be selected by the current MAGMA backend (which matches the coloring). Previously, when the benchmark tuning was done for the MAGMA backend, the MAGMA library would ultimately call cuBLAS for these large sizes, meaning they were mistakenly marked as MAGMA cases when they were "really" cuBLAS cases. The MAGMA backend lookup tables will have to be updated accordingly.

PVC results are very poor for the ported MAGMA routines: the vendor BLAS (oneMKL in this case) is chosen in every case for transpose interpolation at these element sizes. (However, for the non-transpose interpolation, which is not pictured here, there are some smaller cases, with $P = 56$ and $N < 20{,}000$, where a single MAGMA GEMM is chosen over oneMKL.) There is also no trend to favor a single GEMM for the largest problems, found in the bottom-right of the table, unlike the A100 and MI250X cases. On the whole, 20 demonstrates the need for further tuning of the MAGMA routines for SYCL, or perhaps more drastic changes to the source code.

For comparison, we ran these same "stand-alone" benchmarks on A100 (Polaris) and one GCD of MI250X (Crusher). These benchmarks did not perform a sweep to choose the best `nb`, but used the value already selected by the current libCEED backend. In Figure 21, we show the best-performing configuration on the three architectures for the specialized interpolation and gradient kernels, corresponding to first through fourth order tetrahedron elements in MFEM. Figure 22 shows a similar comparison for the standard GEMMs for higher-order elements, with sizes corresponding to fifth through eighth order elements. (Figure 22 only shows interpolation results, as the MAGMA backend currently uses the same GEMM selector decision for gradient,

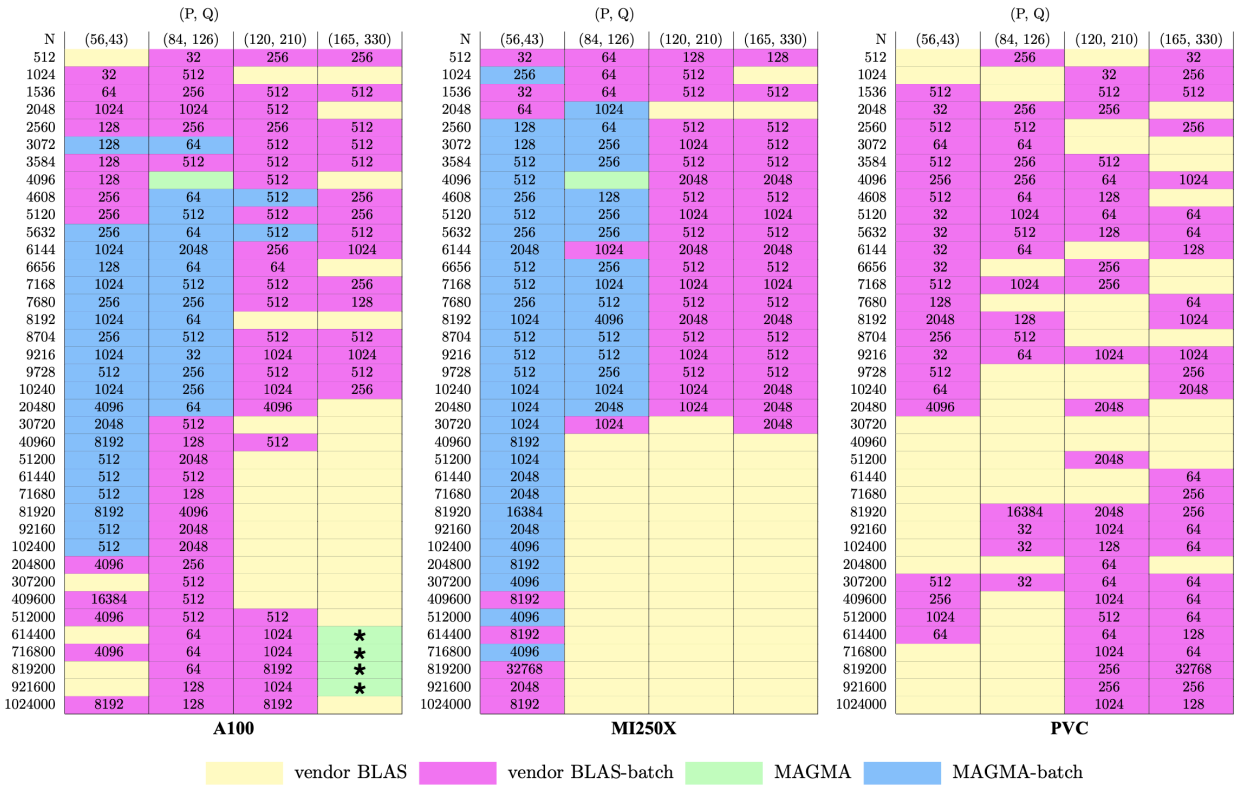

Figure 20: Summary of "GEMM selector" choices for transpose interpolation action on A100, MI250X, and PVC. Text indicates the chosen value of `nb`, the batch size, for a batch GEMM case. The four boxes marked with an * for A100 were swapped from MAGMA to cuBLAS in the results plots due to a change in the MAGMA library, implemented after the benchmark tuning for the current MAGMA backend was done; the backend will be updated accordingly in the future.

with the action repeated three times for a three-dimensional element.) Unfortunately, the "naive" port of the specialized kernels does not perform well in most cases (smaller values of transpose interpolation being the exception). For the standard GEMMs in higher-order elements, we know that the results are always oneMKL, rather than the ported MAGMA code, as shown in Figure 20. The transpose basis action, shown on the left side of Figure 22, has element sizes for which the PVC performance matches or surpasses that of one GCD of the MI250X for large values of N, where the MI250X results suffer a sudden drop. The performance still lags more than expected for the non-transpose basis action; this will need to be investigated further. We note that this case has a "mixed" usage for the input matrices (transpose/non-transpose) in terms of the parameters given to the GEMM routines, while the transpose basis action (in libCEED terminology) uses non-transpose/non-transpose, which is a key difference.

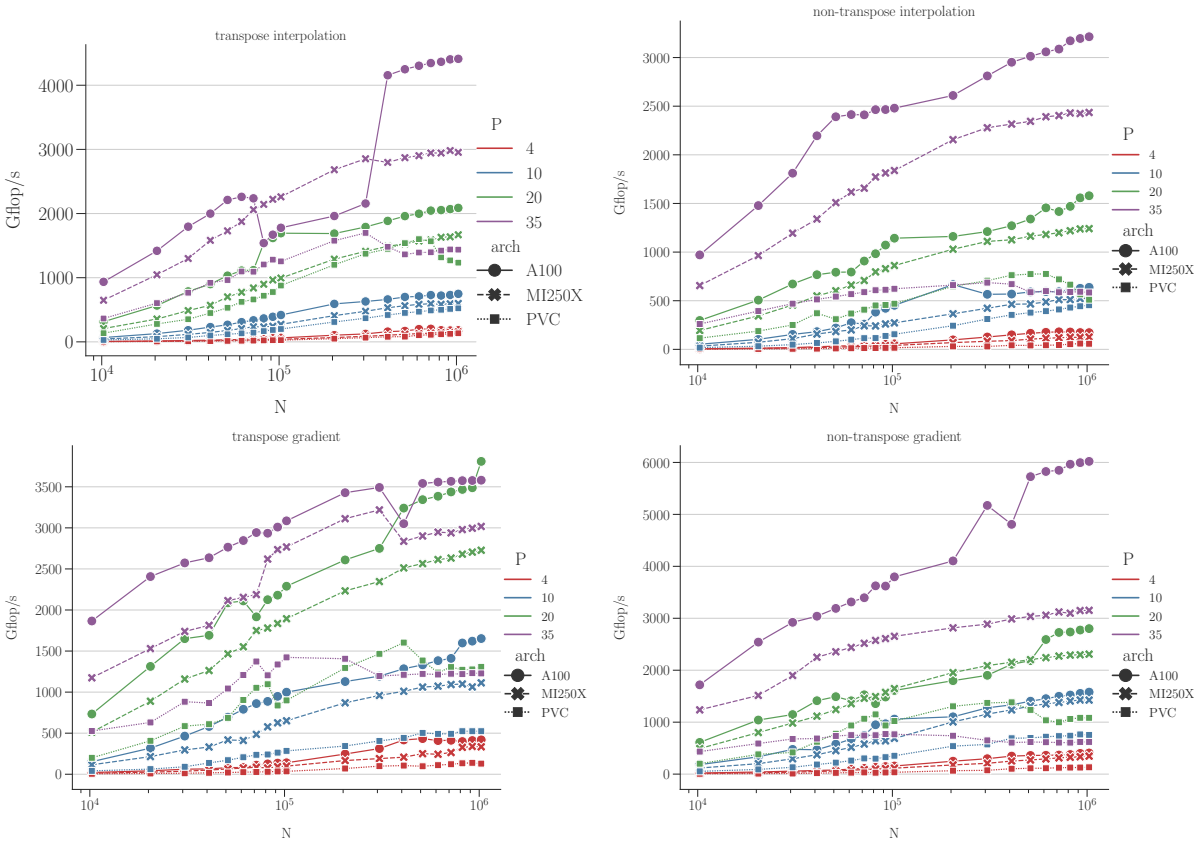

Figure 21: Comparing performance of the specialized MAGMA backend non-tensor interpolation (top row) and gradient (bottom row) kernels. Tests performed on NVIDIA A100 (CUDA 11.4), one GCD of AMD MI250x (ROCm 5.4), and one tile of Intel PVC (oneapi-prgenv/2022.10.15.006.001).

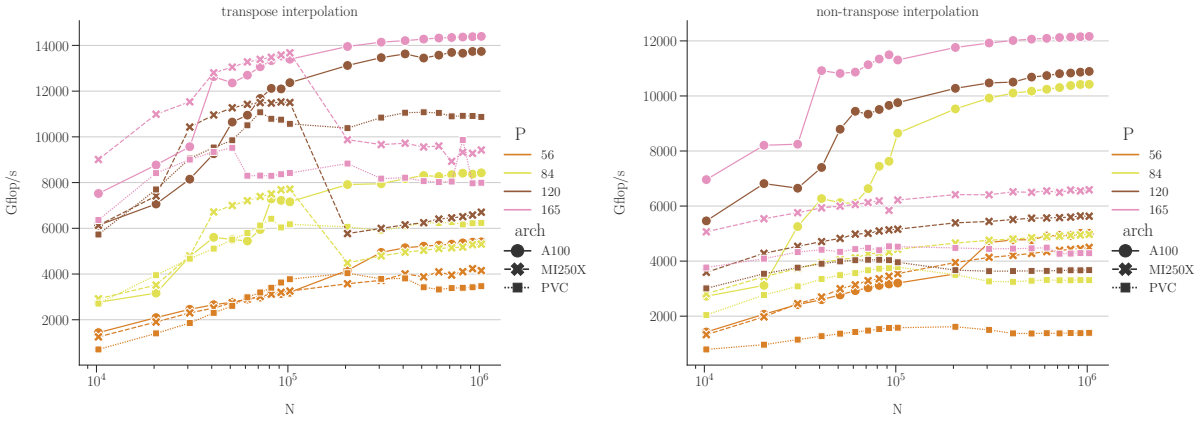

Figure 22: Comparing the standard GEMM/batch GEMM performance for higher-order non-tensor interpolation. Tests performed on NVIDIA A100 (CUDA 11.4), one GCD of AMD MI250x (ROCm 5.4), and one tile of Intel PVC (oneapi-prgenv/2022.10.15.006.001)