ECP Milestone Report

Improve performance and capabilities of CEED-enabled ECP applications on Summit/Sierra

WBS 2.2.6.06, Milestone CEED-MS34

Tzanio Kolev
Paul Fischer
Ahmad Abdelfattah
Shreyas Ananthan
Valeria Barra
Natalie Beams
Ryan Bleile
Jed Brown
Robert Carson
Jean-Sylvain Camier
Matthew Churchfield
Veselin Dobrev
Jack Dongarra
Yohann Dudouit
Ali Karakus
Stefan Kerkemeier
YuHsiang Lan
David Medina
Elia Merzari
Misun Min
Scott Parker
Thilina Ratnayaka
Cameron Smith
Michael Sprague
Thomas Stitt
Jeremy Thompson
Ananias Tomboulides
Stanimire Tomov
Vladimir Tomov
Arturo Vargas
Tim Warburton
Kenneth Weiss

March 30, 2020

ECP Milestone Report
Improve performance and capabilities of CEED-enabled ECP applications on Summit/Sierra
WBS 2.2.6.06, Milestone CEED-MS34

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 30, 2020

ECP Milestone Report
Improve performance and capabilities of CEED-enabled ECP applications on Summit/Sierra
WBS 2.2.6.06, Milestone CEED-MS34

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, 2020	Original	

EXECUTIVE SUMMARY

The goal of this milestone was to help CEED-enabled ECP applications (ExaSMR, MARBL, Urban, ExaWind, ExaAM) improve their performance and capabilities on GPU systems like Summit and Lassen/Sierra. Another goal was to add/improve support for additional hardware and programming models in the CEED software components, release the next version of the CEED software stack, CEED-3.0 and demonstrate performance of libParanumal kernels in libCEED, Nek and MFEM.

In this milestone, the focus was on the following tasks:

- Add/improve support for additional hardware and programming models in the CEED software components;
- Demonstrate performance of libParanumal kernels in libCEED, Nek and MFEM;
- Public release of CEED-3.0;
- Work with CEED applications to improve their GPU performance and capabilities.

A main part of this milestone was adding support for additional hardware and programming models, as well as performance tuning of libParanumal kernels in libCEED, Nek and MFEM. The new developments were released under a CEED-3.0 Public release, and integrated with applications to improve their GPU performance and capabilities.

The artifacts delivered include CEED-3.0 release, performance improvements in applications, tuned CEED software for various architectures through a number of backends, freely available in the CEED's repository on GitHub. See the CEED website, `http://ceed.exascaleproject.org` and the CEED GitHub organization, `http://github.com/ceed` for more details.

In addition to details and results from the above R&D efforts, in this document we are also reporting on other project-wide activities performed in Q2 of FY20 including: ...

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The goal of this milestone was the performance tuning of the CEED software, as well as the use and tuning of CEED to accelerate the first and second wave of targeted ECP applications.

In this milestone, the CEED team developed optimization techniques and tuned for performance the CEED software to accelerate the first and second wave target ECP applications. Specifically, the focus was on the following:

- Efficient use of the memory sub-system for managing data motion and interactions among different physics packages and libraries. This included integration of the new developments in a libCEED 0.5 software release.
- Enhancing the CEED libraries with GPU and AMD GPU support in close interaction with vendors;
- Optimal data locality and motion, and enhanced scalability and parallelism. This included vendors interactions for improvements in data motion and making strong scaling easier and more efficient;
- Continue boosting performance for the first-wave and second-wave of ECP target applications, including ExaSMR, ExaWind, Urban, MARBL, E3SM, and ExaAM.

A main part of this milestone was the performance tuning of the CEED first-wave and second-wave ECP applications. This included the ExaSMR application – Coupled Monte Carlo Neutronics and Fluid Flow Simulation of Small Modular Reactors (ORNL), the MARBL application – Next-Gen Multi-physics Simulation Code (LLNL), and the ExaAM ExaConstit – a miniapp for the Transforming Additive Manufacturing through Exascale Simulation applications project (ExaAM), as well as ExaWind, Urban, and E3SM.

The artifacts delivered include performance improvements in CEED's 1st and 2nd wave of applications, and tuned CEED software for various architectures through a number of backends, freely available in the CEED's repository on GitHub. See the CEED website, `http://ceed.exascaleproject.org` and the CEED GitHub organization, `http://github.com/ceed` for more details.

2. ADDITIONAL HARDWARE AND PROGRAMMING MODELS SUPPORT

2.1 Additional MFEM backends improvements

MFEM version 4.0 introduced new GPU capabilities and support for hardware accelerators such as CUDA, OCCA, libCEED, RAJA and OpenMP. MFEM version 4.1 brings several backend improvements, such as :

- the support for AMD GPUs based on HIP, which is a C++ runtime API and kernel language that can run on both AMD and NVIDIA hardware,
- the improvement of the RAJA backend which offers the same level of performance when targeting NVIDIA's hardware,
- the optimization of multi-GPU MPI communications,
- one special *debug* device specifically designed to aid in debugging GPU code, by following the *device* code path (using separate host/device memory spaces and host $\rightleftarrows$ device transfers) without any GPU hardware.

The MFEM memory manager now also supports different memory types, associated with the following memory backends:

- Default host memory, using standard C++ new and delete,
- CUDA pointers, using cudaMalloc and HIP pointers, using hipMalloc,
- Managed CUDA/HIP memory (UVM), using cudaMallocManaged/hipMallocManaged,

- Umpire-managed memory, including memory pools,
- 32- or 64-byte aligned memory,
- Debug memory with mmap/mprotect protections used by the new *debug* device.

2.2 Porting libParanumal to AMD GPUs

The Frontier system is expected to be online at OLCF in 2021. It will be equipped with purpose built AMD Radeon Instinct GPUs whose specifications that have not yet been unveiled. However it is known that these GPUs will be programmable using the AMD heterogeneous-Commpute Interface for Portability (AMD HIP). It typically takes considerable effort and time to tune up a GPU application so that it fully exploits the hardware and we describe key parts of that process here and provide some results from an intial tuning effort.

Tuning kernels for a typical GPU model first requires us to understand the microarchitecture of the GPU cores (for instance register file and SIMD characteristics), the size of the caches at every level of the memory hierarchy, and the memory access latencies and bandwidths within the hierachy. For NVIDIA GPUs much of this information has been released by NVIDIA or reverse engineered by the micro-benchmrking community for Pascal and Volta series GPUs. Unfortunately this information is not readily available for current generation AMD GPUs and is obviously not available for the GPUs publically released for the 2021 models. A particular challenge when porting from the NVIDIA Volta class to the current generation of AMD GPUs is the limited hardware support for atomic operations, and this necessitates some additional workarounds and code regression to achieve portability from NVIDIA to AMD GPUs.

The second step in tuning GPU kernels is to understand the idiosyncrasies of the GPU kernel compiler. With a decade of experience in using the now mature NVIDIA `nvcc` compiler it is possible to predict how it will transform a given CUDA or OpenCL kernel into GPU binaries. We do not have the same extensive history of working with the relatively new and still rapidly developing AMD HIP compiler `hipcc`. However, recently with the latest releases it has matured signficantly and we are actively working with the AMD Research team to understand the behavior and current best practice for squeezing high performance binaries out of `hipcc`.

The third step in optimizing GPU kernels is refactoring the base kernel implementation to migrate data through the caches of the memory hierachcy to the GPU core registers in an efficient manner while not flooding the limited cache capacities at each level. The eventual goal is to follow Vasily Volkov's advice on GPU optimization and exploit every available cache, while not necessarily being too concerned about occupancy of the GPU cores. This process relies on understanding the compiler code generation tendencies. We have found with `hipcc` that the register usage of generated kernels is highly sensitive to using loop unrolling, much more so than the `nvcc` compiler. With full unrolling we determined that the current `hipcc` compiler is prone to spilling registers to global GPU memory and that occupancy also is heavily impacted. Thus existing CUDA kernels that rely on the way that `nvcc` chooses to unroll loops can perform poorly when compile using HIP and run on AMD GPUs.

The fourth GPU optimization step is to search the space of possible kernels to determine cache blocking parameters, unrolling parameters, and thread work assignments that hit the sweet spot for performance. Fortunately the AMD `hipcc` compiler compilation times have significantly improved enabling comprehensive searches. To build experience we are initially performing this through a manually guided search in kernel space based on a decade of GPU tuning experience. However, it is becoming that with the new HIP compiler and AMD GPU architectures it will be prudent to develop a semi-automated kernel tuning framework. It makes most sense to do that with the CUDA-gen backend developed at LLNL for libCEED.

Finally, in the absence of the actual Frontier 2021 era AMD GPUs we decided to start the tuning warm up process using the currently available AMD Radeon VII GPU. We have unlimited access to locally installed Radeon VII GPUs and they have a hallmark device memory bandwidth of 1TB/s, reasonable FP64 throughput of 3TFLOPS/s, and most importantly can be programmed using AMD HIP or OpenCL. Throughout this process we use a combination of profiling and roofline modeling to determine what the primary performance limiters are and how close to reasonable peak performance each generation of tuned codes can achieve.

We started the tuning process by taking BP1, BP3, and BP5 conjugate gradient solvers implemented in `benchParanumall` using OCCA and tuned for the NVIDIA Volta class GPUs, generating reference timings for the NVIDIA Titan V and then timing them on the AMD Radeon VII. We then used performed a manually

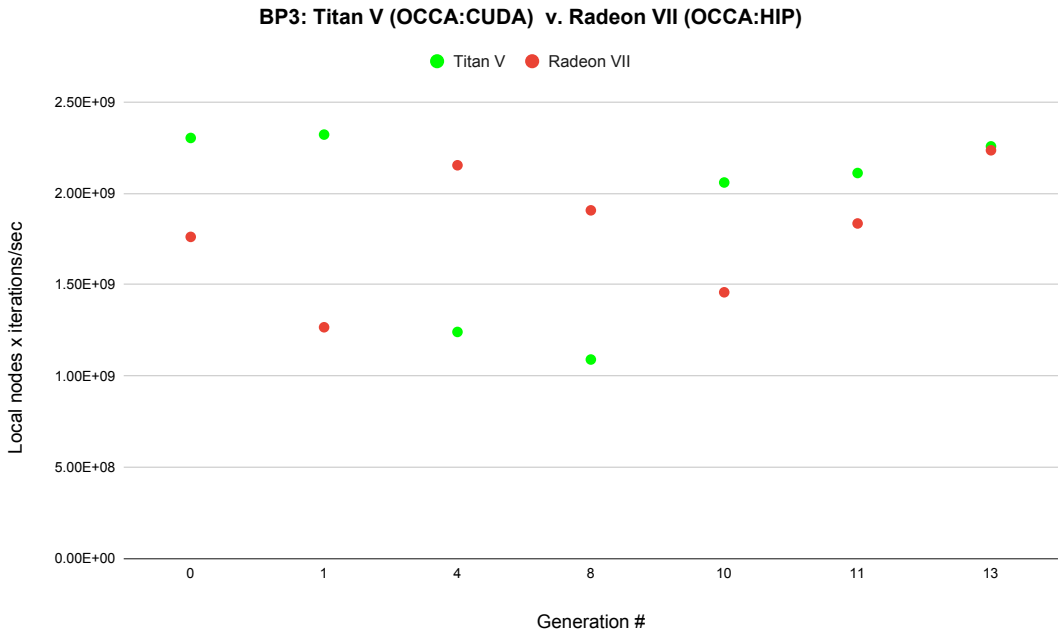

Figure 1: Generations of BP3 implementations tuned to attain high throughput for both the AMD Radeon VII and NVIDIA Titan V GPUs.

directed tuning process to try to find kernel implementations that perform relatively well on both the AMD and the NVIDIA GPU, i.e. we are attempting to develop kernels that are truely performance portable despite different characteristics of the GPUs from the different vendors and their respective tools ecosystems.

In Figure 1 we show the relative performance of different generations of BP3 kernels that are progressively tuned to achieve high performance on both the AMD Radeon VII and the NVIDIA Titan V. To eliminate peripheral issues related to kernel launch latencies we used over 15,000 degree 7 hexahedral elements with a tensor-product arrangement of 9 point Gauss-Legendre rules for integration. The figure illustrates the progression in performance representative generations in the tuning process. We see that the seed code (generation zero) performed better for the Titan V than the Radeon VII GPU as one would expect given that the code had been previously tuned for NVIDIA GPUs. By the fourth generation we see that the performance is reversed as we over corrected for specific details of the AMD GPU. By generation 13 we see that a sweet spot for both GPU models was found and that it does not significantly reduce performance compared to the best achieved GPU performance from prior generations. There is still significant tuning required to achieve roofline limited performance on the AMD Radeon VII. It has a significantly higher device memory bandwidth than the NVIDIA Titan V, but is achieving approximately the same performance for the BP3 benchmark problem. We anticipate that as the AMD HIP compiler matures and as we gain more experience with the AMD GPU that we will be able to further improve performance.

2.3 hipMAGMA

2.3.1 hipMAGMA 1.0

We ported MAGMA to HIP and made an initial hipMAGMA version 1.0 release on March 9, 2020. hipMAGMA is based on MAGMA 2.5.2 and provides the MAGMA 2.5.2 functionalities to AMD GPUs. All CUDA-based sources in MAGMA 2.5.2 are converted to HIP. Installation and scripts to do the conversion were developed and provided with the release. Thus, there is a single source to maintain, that is currently CUDA-based, and other backends are automatically generated. The library can be installed to support either NVIDIA GPUs or AMD GPUs through a single interface. We note that the effort to add support for AMD GPUs

was fairly light. Currently, we have complete functional port for the entire MAGMA library. Performance portability is derived from the use of the BLAS standard and its optimized implementations, either from vendors or other open source libraries, like MAGMA. The performance portability and current performance is illustrated in Secions 2.3.2 and 2.3.3 for general dense linear algebra workloads and CEED-specific batched tensor contractions.

Architecture-specific optimizations are typically needed for dense linear algebra. Besides using BLAS, we included specific optimizations and tuning for some of the BLAS routines. We also developed and released a number of BLAS kernels that are not available in hipBLAS yet. This included SYMM, SYRK, and SYR2K for all IEEE floating-point arithmetic precisions. The developments were done through CUDA and released through MAGMA 2.5.3 on March 29, 2020. MAGMA 2.5.3 is also integrated in the CEED 3.0 release.

2.3.2 Enabling MAGMA for AMD GPUs based on the HIP Runtime

The MAGMA backend for libCEED is hybrid in the sense that it relies on both customized kernels that are solely developed for libCEED, as well as existing linear algebra kernels (BLAS) that are already part of the MAGMA library. The first step towards supporting modern AMD GPUs is to build an interface for the `HIP` runtime inside MAGMA. This abstraction of runtime and vendor-supplied math libraries seamlessly allows MAGMA to run on top of the `CUDA` and the `HIP` runtimes. The initial release of hipMAGMA enables all of the runtime functionalities required for libCEED (e.g. memory management and data transfers). It also provides most of the compute-bound BLAS kernels that are critical to the non-tensor basis action in libCEED backends.

In order to assess the quality of the "hipMAGMA backend" for libCEED, benchmarks for standard linear algebra kernels can give helpful insights about how far we can push AMD GPUs for both memory-bound and compute-bound kernels. In fact, the standard matrix multiplication (`dgemm`) and its batched variant (`dgemm_batched`) are used in the non-tensor basis actions in libCEED. The performance of these kernels inside libCEED is better to be put in perspective by comparing it to performance tests that are designed to test the achievable compute power and memory bandwidth of AMD GPUs. Figure 2 illustrates the performance of a

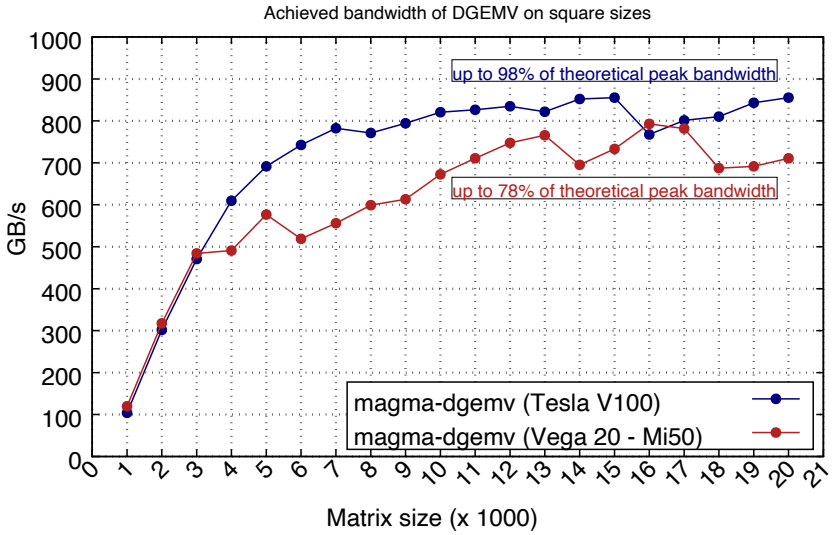

Figure 2: The matrix-vector multiply benchmark in double precision (DGEMV). Results are shown for the Tesla V100 GPU (CUDA 9.1) and the Mi50 GPU (HIP 3.0).

memory-bound kernel, the standard matrix-vector multiply operation in BLAS. On each GPU, the results combine the best performance observed from the vendor BLAS (cuBLAS or hipBLAS) and the MAGMA BLAS kernels. On the Tesla V100 GPU, the performance is very close to the theoretical peak bandwidth (93% out of 900 GB/s). On the Mi50 GPU, the best recorded performance is about 78% of the peak bandwidth (1024 GB/s). Although the Mi50 GPU has a higher theoretical bandwidth, the achievable performance is less

than that on the V100 GPU. This can be due to the fact that several technologies for AMD GPUs (hardware, compiler, runtime, and others) are not as mature as the NVIDIA GPUs and the CUDA Runtime.

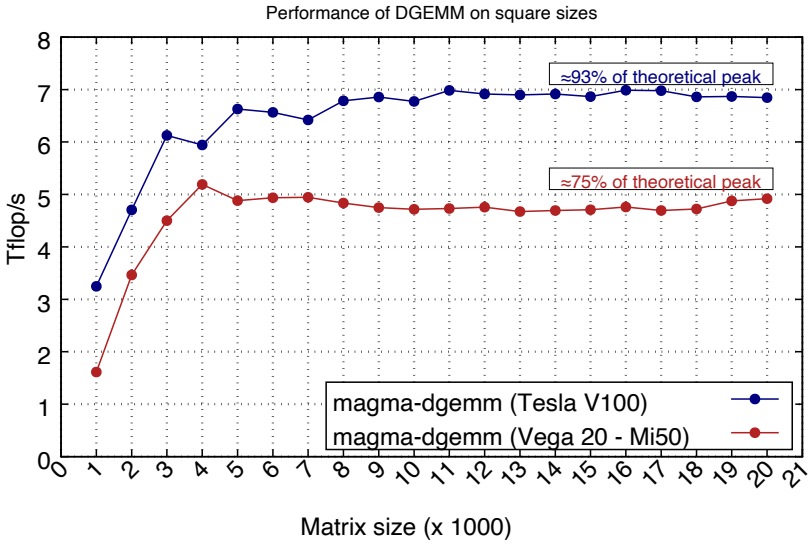

Figure 3: The matrix-matrix multiply benchmark in double precision (DGEMM). Results are shown for the Tesla V100 GPU (CUDA 9.1) and the Mi50 GPU (HIP 3.0).

Figure 3 shows a compute-bound benchmark; the matrix-matrix multiplication kernel (DGEMM). Similar to the DGEMV benchmark, the achievable DGEMM performance on the V100 GPU is significantly better than that on the Mi50. Although the Mi50 GPU has a theoretical peak performance of 6.6 Tflop/s, the DGEMM performance stagnates at ≈ 5 Tflop/s. For this specific benchmark, both cuBLAS and hipBLAS outperform the kernels available in MAGMA. However, we observe that the MAGMA kernels usually perform better on NVIDIA GPUs than on AMD GPUs, even when tuning is taken into account. As an example, the MAGMA DGEMM kernel reaches up to 5.9 Tflop/s of performance on the V100 GPU (84.3% of the best cuBLAS performance). The same code was "hipified" and tuned for the Mi50 GPU, but no more than 2.5 Tflop/s was achieved (51% of the hipBLAS performance). Our current experience with the `hipcc` compiler shows that maybe a different set of optimizations/practices have to be considered for AMD GPUs. It is also possible that the `hipcc` compiler will drastically evolve in future releases, as it is a fairly recent compiler.

The reason behind benchmarking the batched DGEMM kernel is that it can outperform the regular DGEMM kernel for some use cases in the non-tensor basis action of the MAGMA backend, which we highlight in Section 2.3.3. Figure 4 shows the performance of the batched DGEMM operation on both GPUs for square sizes. The performance behavior is similar, with a slightly lower percentage for the Mi50 GPU, and less stable performance on both GPUs. However, it is interesting to point out that the MAGMA kernels were able to outperform the vendor-BLAS for relatively small sizes. On the Tesla V100 GPU, the MAGMA kernels were better than cuBLAS for sizes less than 100. On the Mi50, the MAGMA kernels outperformed hipBLAS for sizes less than 400.

2.3.3 BLAS Performance for Non-tensor Basis Action in libCEED

The non-tensor basis action in libCEED can be performed using the General Matrix-Matrix Multiplication kernel (DGEMM). The standard DGEMM operation is defined as an update to a Matrix C such that: $C_{M\times N} = \alpha A_{M\times K} \times B_{K\times N} + \beta C_{M\times N}$. In terms of the call configuration inside libCEED, the DGEMM operation may involve a transposition of A.

Considering the common sizes in the libCEED bake-off problems, the values of M and K are relatively small, compared to N. In fact, $N = nelem \times ncomp$, where $nelem$ is the number of elements and $ncomp$ is the number of components. The shape of the DGEMM operation is shown in Figure 5. Ideally, a single DGEMM operation would be enough to operate at the GPU peak performance. However, the relatively large

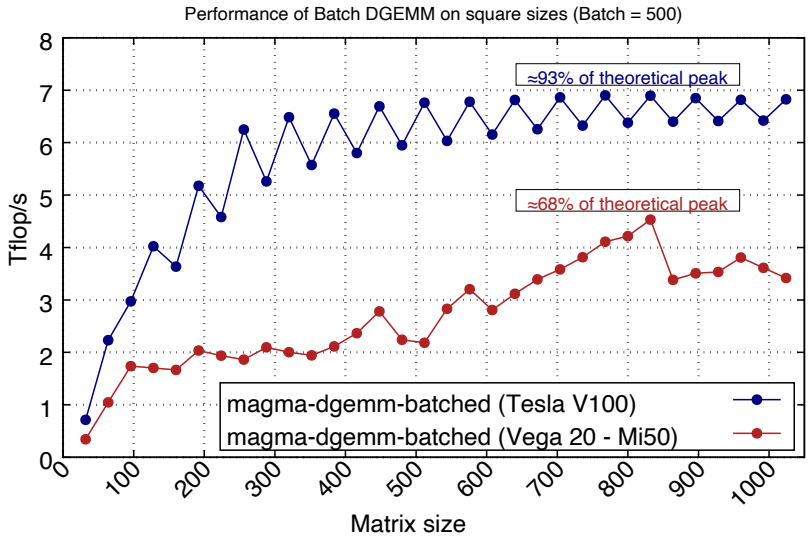

Figure 4: The batched matrix-matrix multiply benchmark in double precision (DGEMM). Results are shown for the Tesla V100 GPU (CUDA 9.1) and the Mi50 GPU (HIP 3.0).

value of N can be broken down into smaller sizes that are independently processed as a batch, hence using batched DGEMM is also a viable option in the MAGMA backend.

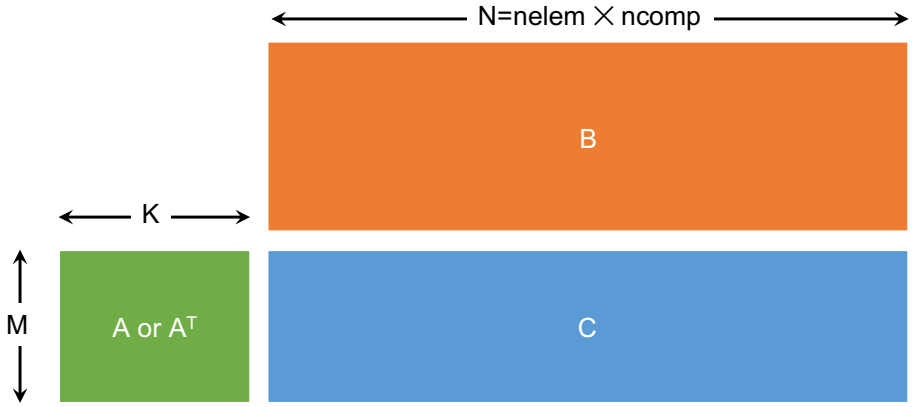

Figure 5: Shape of the DGEMM operation for the non-tensor basis action in libCEED.

Figure 6 shows the performance of DGEMM and its batch variant for three typical problem sizes in the non-tensor basis action in libCEED. The figure shows the performance for the V100 and the Mi50 GPUs. The first observation is that relatively larger problems can be compute bound as they achieve around 80% of the DGEMM peak on the same hardware. Another observation is that the batch DGEMM can be quite competitive with the regular DGEMM, which is the case for the V100 GPU, where the batch DGEMM performance is better than or equal to the regular DGEMM performance. On the Mi50, however, the performance inconsistency of its batch DGEMM results in some winning scenarios for the regular DGEMM kernel. The MAGMA backend will provide tuning mechanisms to decide the kernel to be invoked and the best tuning parameters for the given problem size.

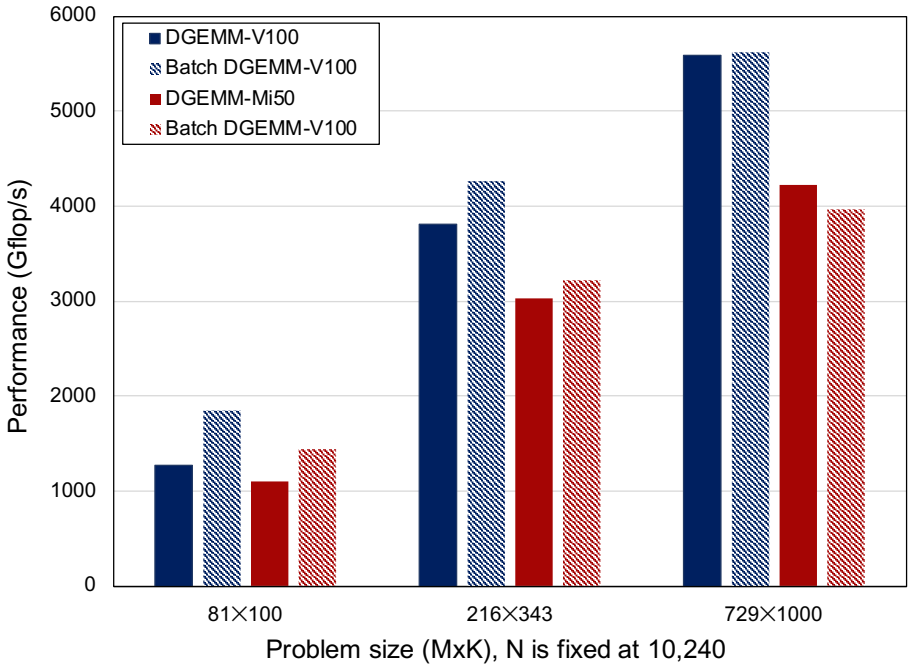

Figure 6: Performance for the non-tensor basis action in libCEED for typical problem sizes using DGEMM and Batch DGEMM.

2.3.4 Status of Runtime Compilation for HIP

Currently, the CUDA and MAGMA backends make use of `nvrtc` for runtime compilation. Many of `libCEED`'s runtime parameters, such as the order of the basis functions and the number of quadrature points, should actually be known at compile time in order to produce efficient GPU code through the use of register memory and the possibility for more compiler optimizations. To this end, the CUDA backends use runtime compilation with `nvrtc` to produce the kernels for element restrictions, basis actions, and quadrature functions (QFunctions) – or, in the case of the CUDA-gen backend, to produce the fused operator kernel. The MAGMA backend, on the other hand, takes the approach of using kernel templates. However, as the QFunction is allowed to be defined by the user in a single C-style source file, some sort of runtime compilation is still necessary to be able to turn a user-specified function into GPU code. Indeed, the current MAGMA backend reuses the QFunction action of the CUDA-ref backend.

A `hiprtc` tool, meant as a HIP replacement for `nvrtc`, has been available in HIP since May 2019, with updates coming in late 2019 and early 2020. Lack of official documentation or information about its use left its ability to replace `nvrtc` in `libCEED` backends as a crucial but open question for HIP backend developement. Recently, the CEED MAGMA team began working on porting the `nvrtc`-related code from the CUDA-ref backend in order to test the current status of `hiprtc`. An experimental "HIP-ref" backend has been created with fairly minor changes related to replacing `nvrtc`, such as adding an include statement for the HIP runtime header in the source of the compiler calls and modifying the names of single-character parameters passed to the compiler as macro definitions. There is the potential for more disruptive changes required due to `hiprtc` using features not included in HIP's current C-only header file, but these challenges, if they occur, should be surmountable. As the use of `hiprtc` for the QFunction action has been demonstrated and hipMAGMA is released, the HIP version of the current MAGMA backend is now definitively possible. CUDA-gen-style code generation may also be possible for HIP through `hiprtc`. The performance of `hiprtc` in terms of compilation speed and quality of code produced is still unknown; however, as `hiprtc` is still under active development, it can reasonably be expected to improve if it is found to be significantly inferior to `nvrtc`.

2.4 Intel GPUs and SYCL

3. PERFORMANCE OF LIBPARANUMAL KERNELS IN LIBCEED, NEK, AND MFEM

The CEED `libParanumal` library is an experimental testbed for exploring plugin GPU-capable components that can be integrated into existing high-order finite element codes as optional accelerator modules. This library is being actively developed by the CEED team at Virginia Tech. The goal of this milestone is to take the highly optimized kernels from `libParanumal` and make them more widely available to CEED applications, by incorporating those kernels in the CEED low-level API library (`libCEED`) and the CEED high-level API codes (`MFEM` and `Nek5000`)

CEED-T2 Sub-tasks (see ECP Jira): Items marked with ✓are considered to be completed or equivalent outcomes have been achieved.

- ✓ Integrate `libParanumal` kernels in `Nek5000` (see Section 3.4).
- ✓ Integrate `libParanumal` kernels in `MFEM` (see Section 3.2).
- ✓ Integrate `libParanumal` kernels in `libCEED` (see Section 3.3).
- ✓ Demonstrate improved performance with the new `libParanumal` in at least two of the above CEED software components (see Sections 3.4,3.2,and 3.3)
- ✓ Develop and document a process to ease the integration of new `libParanumal` kernels in `libCEED`, `Nek5000` and `MFEM` in the future (see Section 3.5).

Completion criteria The following completion criteria are described and documented within this report.

- ✓ Support for `libParanumal` kernels in the master branches of at least two CEED software components (`libCEED`, `Nek5000` and/or `MFEM`).
- ✓ Documented performance improvements with the new `libParanumal` to be included in the CEED-MS34 (ADCD04-53) report.

3.1 Introduction

The CEED VT team developed a set of highly optimized implementations for the CEED bake-off kernels BK1-6 and the associated bake-off problems BP1-6. These bake-off benchmarks encompass several core finite element operations including evaluating the action of a stiffness matrix on a vector of unknowns. Implementations of these kernels are featured in the recently released `benchParanumal` benchmark codes that were spun off from the `libParanumal` library to facilitate rapid prototyping of kernels. `benchParanumal` includes reference implementations in the CUDA, HIP, and OCCA programming models. The first goal of the CEED-T2 milestone task is to analyze, scrutinize, adapt and incorporate critical features from the highly optimized `benchParanumal` kernels into the `libCEED` library of portable finite element operator implementations and the LLNL `MFEM` modular finite element project.

This activity was facilitated in part by Yohann Dudouit visiting Virginia Tech for a week following the 3rd CEED Annual Meeting. A comprehensive comparison of the existing `libCEED` and `benchParanumal` compute kernels was used to determine a plan for how to modify the output of the CUDA-gen backend of `libCEED` to obtain similar performance to `benchParanumal`.

The VT team has also developed a showcase library called `libParanumal` that is built on top of OCCA kernels from `benchParanumal`. The `libParanumal` library includes example mini-apps include high-order finite element solvers for elliptic problems, compressible Navier-Stokes, incompressible Navier-Stokes, and Boltzmannian gas dynamics. The second goal in the CEED-T2 milestone task is to incorporate the `libParanumal` elliptic solvers and other capabilities into the `NekRS` portable high order spectral element solvers for incompressible flow simulations being developed by the CEED UIUC and ANL teams.

3.2 Integrating `libParanumal` kernels in `MFEM`

Based on the algorithms from benchParanumal, the `MFEM` team has developed a pathway to integrate `libParanumal` kernels directly in `MFEM`. This is in addition to the CUDA kernels in `libCEED` that were optimized in collaboration with `libParanumal` and are also available in `MFEM` via the `libCEED` integration, see Section 3.3.

The pathway to `libParanumal` kernels integration is based either on the OCCA versions of the bench-Paranumal kernels (`MFEM` already supports OCCA) or on the device versions of the kernels based on new features, such as mapping data to registers, that have been added in a development branch of `MFEM`. Another important feature is loop unrolling in `MFEM_FORALL` kernels, which will require additional developments in `MFEM` in order to achieve the same level of performance. Exclusive memory was added to the `MFEM` 's kernels by providing an extra token, `MFEM_REGISTER`, to allow the addition of such qualifier to local variables. The concept of exclusive memory is similar to thread-local storage: one variable gets its own private value per thread. This new keyword makes it possible to use `libParanumal`'s kernels by embedding them into lambda-captured `MFEM_FORALL` kernels. In Listing 1 we show how OCCA kernel code from `libParanumal` can now be easily translated into the lambda capture approach used by `MFEM`.

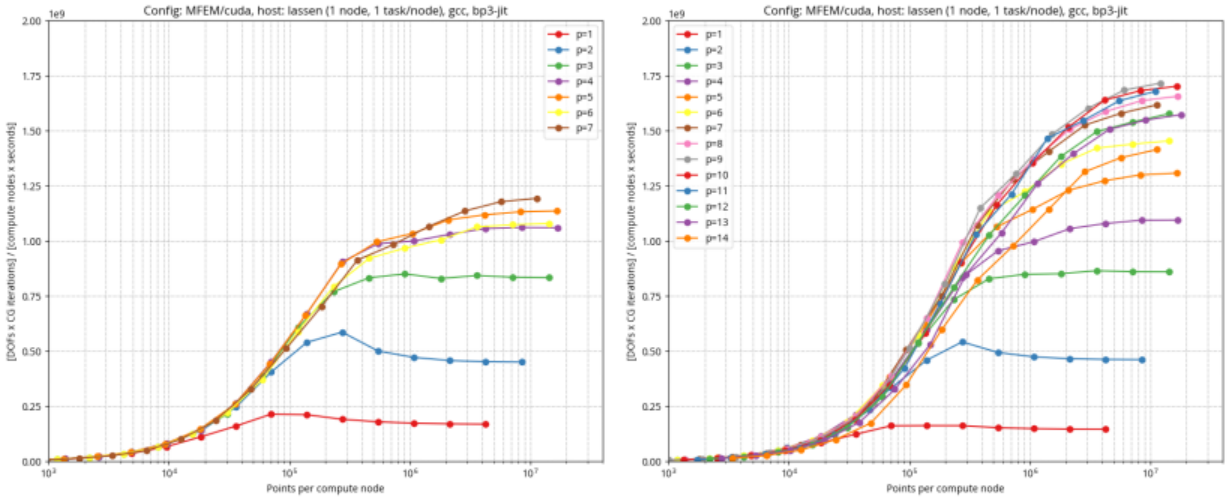

Figure 7: Left: original `MFEM` diffusion kernel. Right: `libParanumal` style kernels in `MFEM`.

The `MFEM` speed up results after the diffusion kernel was transformed in this way are shown in Figure 7. We see that performance was improved for the diffusion kernel with polynomial degrees of 4 and higher and that the new `libParanumal` variant kernels are able to be used up to degree 14 hexahedral elements whereas the `MFEM` variants were only able to work up to degree 7. The speed ups and increased degree capability are due to the careful tuning of the `libParanumal` kernel implementation, unrolling, cache management, and the two-dimensional slicing algorithm imported from `libParanumal` [16]. This algorithm and tuned implementations were the result of many generations of kernel optimization and refinement and is able to compute the action of the finite element diffusion operator at the streaming rate of the state vector, output vector, and geometric information of the element.

The new `MFEM` infrastructure developed to import the `libParanumal` algorithms also enables the generalization from the canonical CEED bake-off problems handled by `libParanumal` to the more general capabilities of `MFEM` while also preserving highly tuned performance. `MFEM` now has unique performance capabilities for general high-order finite element calculations on GPUs.

```
@kernel void BP3Global_v0(const dlong
    Nelements,
  @restrict const dlong *elementList,
  @restrict const dlong *localizedIds,
  @restrict const dfloat *ggeo,
  @restrict const dfloat *D,
  @restrict const dfloat *I,
  const dfloat lambda,
  @restrict const dfloat *q,
  @restrict dfloat *Aq){

  for(dlong e=0; e<Nelements; ++e; @outer(0)){

    @shared dfloat s_Iq[p_cubNq][p_cubNq][
    p_cubNq];
    @shared dfloat s_D[p_cubNq][p_cubNq];
    @shared dfloat s_I[p_cubNq][p_Nq];
    @shared dfloat s_Gqr[p_cubNq][p_cubNq];
    @shared dfloat s_Gqs[p_cubNq][p_cubNq];

    @exclusive dfloat r_qt;
    @exclusive dfloat r_q[p_cubNq], r_Aq[
    p_cubNq];
    @exclusive dlong element;

    for(int j=0;j<p_cubNq;++j;@inner(1)) {
      for(int i=0;i<p_cubNq;++i;@inner(0)) {
        s_D[j][i] = D[p_cubNq*j+i];
        if(i<p_Nq) { s_I[j][i] = I[p_Nq*j+i];
    }
  ...

```

```
template<int T_D1D = 0, int T_Q1D = 0>
void BP3Global_v0(const int NE,
                  const Array<double> &B,
                  const Array<double> &G,
                  const Vector &D,
                  const Vector &X,
                  Vector &Y,
                  const int d1d = 0,
                  const int q1d = 0)
{
   const int D1D = T_D1D ? T_D1D : d1d;
   const int Q1D = T_Q1D ? T_Q1D : q1d;
   constexpr int MD1 = T_D1D ? T_D1D : MAX_D1D
    ;
   constexpr int MQ1 = T_Q1D ? T_Q1D : MAX_Q1D
    ;

   auto b = Reshape(B.Read(), Q1D, D1D);
   auto g = Reshape(G.Read(), Q1D, Q1D);
   auto d = Reshape(D.Read(), Q1D*Q1D*Q1D, 6,
    NE);
   auto x = Reshape(X.Read(), D1D, D1D, D1D,
    NE);
   auto y = Reshape(Y.ReadWrite(), D1D, D1D,
    D1D, NE);

   MFEM_FORALL_2D(e, NE, Q1D, Q1D, 1,
   {
      MFEM_SHARED double s_Iq[MQ1][MQ1][MQ1];
      MFEM_SHARED double s_D[MQ1][MQ1];
      MFEM_SHARED double s_I[MQ1][MD1];
      MFEM_SHARED double s_Gqr[MQ1][MQ1];
      MFEM_SHARED double s_Gqs[MQ1][MQ1];

      double MFEM_REGISTER_2D(r_qt,MQ1,MQ1);
      double MFEM_REGISTER_2D(r_q,MQ1,MQ1)[MQ1
    ];
      double MFEM_REGISTER_2D(r_Aq,MQ1,MQ1)[
    MQ1];

      MFEM_FOREACH_THREAD(j,y,Q1D) {
        MFEM_FOREACH_THREAD(i,x,Q1D) {
            s_D[j][i] = g(i,j);
            if (i<D1D) { s_I[j][i] = b(j,i); }
            if (i<D1D && j<D1D) {
               for (int k = 0; k < D1D; k++) {
                  MFEM_REGISTER_2D(r_q,j,i)[k]
                  = x(i,j,k,e);
      ...
```

Figure 8: Example translation of `libParanumal` OCCA kernel code (left) into `MFEM` lambda code (right) using `MFEM_FORALL` with the `MFEM_REGISTER` qualifier.

3.3 Integrating libParanumal kernels in libCEED

Contrary to MFEM and Nek5000, initially libCEED did not have a library of available operators. Operators could only be described at runtime by the user, preventing operator specific optimizations. However, the recent development of the QFunction gallery in libCEED allows to target specific known operators, e.g. BP1, BP3, enabling the direct integration in libCEED of highly optimized kernels targeting specific operators. This new feature offers a clear path for the integration of libParanumal kernels in libCEED, and an effort in this direction is in libCEED's roadmap. Nevertheless, the experience gathered by the CEED team developing libParanumal at VT was used to substantially improve the performance of libCEED (See Figure 9 for a comparison before/after on BP3 using libCEED through MFEM).

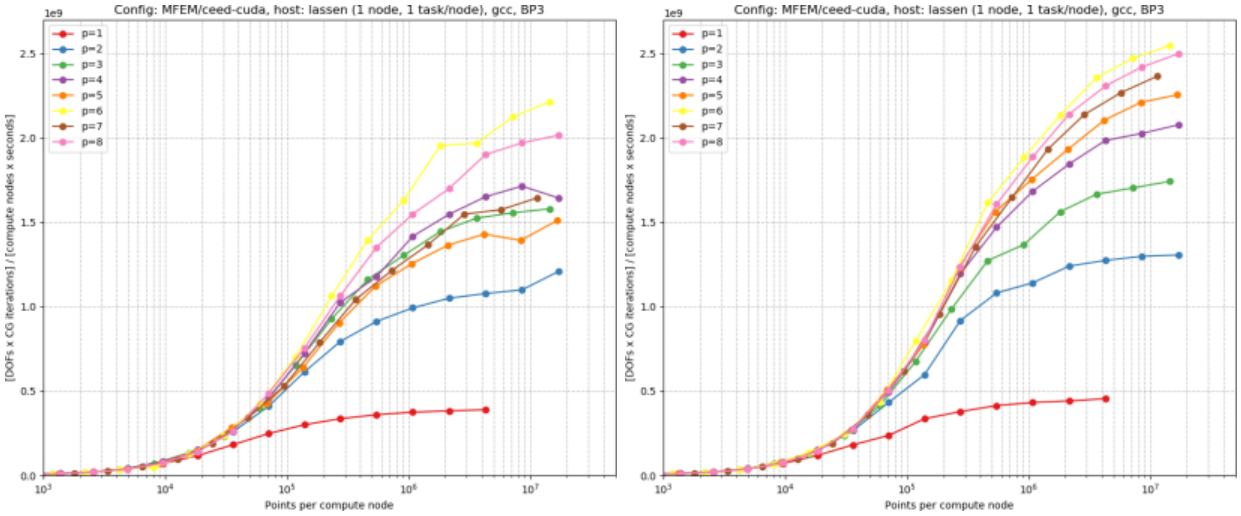

Figure 9: Left: original libCEED diffusion kernel. Right: libCEED diffusion kernel after using libParanumal optimizations.

During a workshop at VT, analyzing the differences between libParanumal kernels for BP1 and BP3, and the code generated by the cuda-gen backend of libCEED revealed the causes of the main performance gaps. Transferring the identified libParanumal's critical code optimizations to the cuda-gen backend almost closed the performance gap between libParanumal and libCEED for BP1 and BP3. In addition, these optimizations also resulted in an overall performance improvement for all operators generated by the cuda-gen backend of libCEED. However, libParanumal still has an edge over libCEED on other BP problems, therefore future workshops are planned to further improve libCEED's code generation using the work done in libParanumal. It should be emphasized that one of the most positive aspect of this workflow is that the highly optimized kernels developed in libParanumal, targeting specific operators, result in an overall performance improvement for libCEED over the whole spectrum of possible operators defined arbitrarily by the user.

3.4 Integrating libParanumal into NekRS

NekRS is a new C++ variant of Nek5000 based on the OCCA project and the libParanumal library developed by Warburton's group at Virginia Tech as part of the CEED project since 2017 as a test platform for exploring advanced algorithms for PDEs. NekRS development started in January, 2019 with the goal of reproducing the operational capabilities of Nek5000 including additional support for portable heterogeneous computing focusing on GPU acceleration.

Figure 10 demonstrates turbulent flow in 17x17 rod-bundle computed with NekRS on Summit. Figures 11 shows our baseline performance results for the very initial version of NekRS, released on GitHub in November 2019, performed on Summit for the 17x17 rod-bundle flow simulation illustrated in Figure 10. The mesh uses 277000 elements of order $N = 7$ ($n = 95M$ gridpoints total). The Reynolds number is 5000 based on hydraulic diameter. Periodic boundary conditions are used in the axial flow direction and the initial conditions comprise an array of meandering vortices.

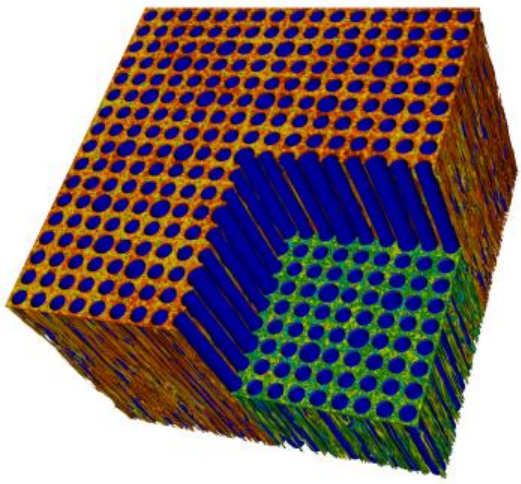

Figure 10: `NekRS` 17x17 rod-bundle turbulent flow simulation performed on the OLCF Summit system.

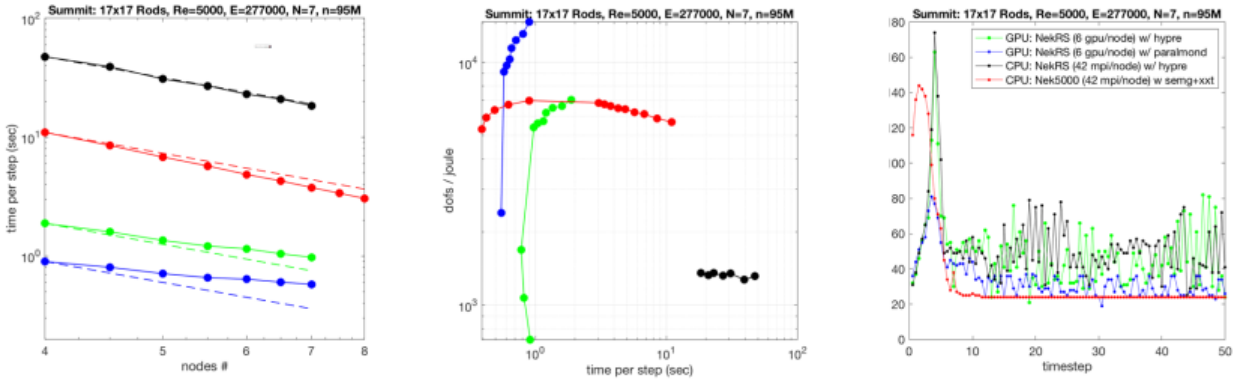

Figure 11: `NekRS`S and `Nek5000` performance of GPUs vs. CPUs on Summit for turbulent flow simulations with Re=5000 for a 17x17 rod-bundle geometry using total number of grid points n=95,011,000. Based on timings from Step 11 to 60, time-per-step with ideal scalings shown as dashed lines (left), pressure iterations per step (center), and dofs-per-joule with respect to time-per-step (right) are shown.

Figure 11, left, shows strong scaling results on a few nodes of Summit using `NekRS` with six V100 GPUs per node or `NekRS`/`Nek5000` with 42 CPU cores per node. For the CPU version, `NekRS` uses Hypre as a coarse grid solver. In this case, `NekRS` running on the CPUs is about 4X slower than `Nek5000` because the pressure solver is not yet as optimized as the highly-tuned solver in `Nek5000`. For the GPU, the `NekRS` results improve substantially when the coarse grid solver is based on the AMG solver ParAlmond developed by Warburton's reseach group.

Figure 11, center, shows the pressure iteration counts for each of the four cases. `Nek5000` uses Schwarz-smoothed p-multigrid while `NekRS` uses Chebyshev smoothing. When ParAlmond is used for the coarse-grid solve the `NekRS` iteration counts improve by a factor of two and are on par with those of `Nek5000`. However, the Chebyshev smoother requires more work per iteration than the Schwarz-based smoother.

With ongoing effort on the pressure solve we anticipate a 2X reduction in `NekRS` solution times, which will put it on par with the strong-scaled solution times of `Nek5000` with more than 2X energy savings that are already observed for `NekRS` on Summit's V100s (Figure 11, right).

3.5 Related portability activities

The CEED team is actively engaged with Intel in preparation for the Intel-based Aurora architecture. Activities in this area include: performance projection for Nekbone on the new Aurora architecture, estimated to achieve over 50x FOM speedup over Sequoia's baseline; attending the Aurora workshop at ANL (9/17-9/19, 2019) by 5 CEED members; initial porting of `NekRS` to an Aurora development system, running `NekRS` in OpenCL mode (via OCCA) on a single Intel Gen9 GPU; and further kernels optimization of OCCA, libCEED, `libParanumal`, and `Nek5000` planned and/or in development.

CEED researchers are also actively engaged with porting to and evaluation of AMD GPUs in preparation for the AMD-based Frontier machine. Activities in this area include: support for HIP in OCCA and MFEM-4.0; initial MFEM performance runs on the LLNL Corona cluster, which has Radeon Instinct MI25 GPUs (weak double precision); initial results with `libParanumal` on Radeon VII (good double precision) which on certain benchmarks have achieved over 75% of the NVIDIA V100 peak performance at 10% of the cost; and collaboration with AMD on fixing slow linking times with the HIP compiler which was reported to hamper development at the CEED annual meeting, and was addressed by AMD engineers within a couple of weeks.

Table 1 demonstrates `NekRS` baseline of performance measured on a single GPU on V100 and Intel Gen9. Simulations are performed for turbulent flows with a Reynolds number of 8000 using triangle-shaped pipe geometry with 9234 elements of order $N = 7$ ($n = 3M$ gridpoints total). Wall boundary in spanwise and periodic boundary in stream directions are considered with turbulent initial condition. Timings are measured from 100 timestep runs.

Table 1: NekRS baseline of performance measure on a single GPU, Intel Gen9 (Aurora development system) vs. Nvidia V100. Turbulent pipe simulations with $Re = 8000$ for 100 timestep runs with $E = 9234, N = 7, n = EN^3 = 3,167,262$.

systems	API backend	100 steps	time per step	ratio
Intel Gen9 (Iris@JLSE/ANL)	OpenCL	1.78498e+03 (sec)	17.84 (sec)	1
Nvidia V100 (Nurburg@ANL)	OpenCL	3.88553e+01 (sec)	0.388 (sec)	45.93
Nvidia V100 (Nurburg@ANL)	CUDA	3.75509e+01 (sec)	0.375 (sec)	47.53
Nvidia V100 (Summit@OLCF)	CUDA	3.83653e+01 (sec)	0.386 (sec)	46.53

3.6 On-going integration of `benchParanumal` and `libParanumal` kernels into CEED packages

The last component of the CEED-T2 task was to create an improved process to incorporate ongoing and future developments from `benchParanumal` and `libParanumal` into the CEED `Nek5000`, `MFEM`, and `libCEED` packages. This has been addressed in the following ways.

- The fastest changing part of `libParanumal`, namely the CEED bake-off kernels and CEED bake-off problem codes was split off into a separate project `benchParanumal`. This enabled the CEED ANL and UIUC terms to build `NekRS` on top of `libParanumal` as the latter now changes less frequently.

- Work is already under way to tune `benchParanumal` kernels for AMD GPUs using the OCCA HIP backend. Kernels from `benchParanumal` will be migrated into `libParanumal` and consequently up to `NekRS` as they are tested and proved capable for the AMD GPUs on Frontier and the Intel GPUs on Aurora.
- The `MFEM` CUDA-gen backend was extended to capture more capabilities exploited by the kernels in `benchParanumal`. Incorporating future kernels from `benchParanumal` into `MFEM` will thus be more straight forward going forward.

4. LIBCEED-0.6 RELEASE

libCEED v0.6 contains numerous new features and examples, as well as expanded documentation in this new website.

4.0.1 New features

- New Python interface using CFFI provides a nearly 1-1 correspondence with the C interface, plus some convenience features. For instance, data stored in the `CeedVector` structure are available without copy as `numpy.ndarray`. Short tutorials are provided in Binder.
- Linear QFunctions can be assembled as block-diagonal matrices (per quadrature point, `CeedOperatorAssembleLinearQFunction`) or to evaluate the diagonal (`CeedOperatorAssembleLinearDiagonal`). These operations are useful for preconditioning ingredients and are used in the libCEED's multigrid examples.
- The inverse of separable operators can be obtained using `CeedOperatorCreateFDMElementInverse` and applied with `CeedOperatorApply`. This is a useful preconditioning ingredient, especially for Laplacians and related operators.
- New functions: `CeedVectorNorm`, `CeedOperatorApplyAdd`, `CeedQFunctionView`, `CeedOperatorView`.
- Make public accessors for various attributes to facilitate writing composable code.
- New backend: `/cpu/self/memcheck/serial`.
- QFunctions using variable-length array (VLA) pointer constructs can be used with CUDA backends. (Single source is coming soon for OCCA backends.)
- Fix some missing edge cases in CUDA backend.

4.0.2 Performance Improvements

- MAGMA backend performance optimization and non-tensor bases.
- No-copy optimization in `CeedOperatorApply`.

4.0.3 Interface changes

- Replace `CeedElemRestrictionCreateIdentity` and `CeedElemRestrictionCreateBlocked` with more flexible `CeedElemRestrictionCreateStrided` and `CeedElemRestrictionCreateBlockedStrided`.
- Add arguments to `CeedQFunctionCreateIdentity`.
- Replace ambiguous uses of `CeedTransposeMode` for L-vector identification with `CeedInterlaceMode`. This is now an attribute of the `CeedElemRestriction` (see `CeedElemRestrictionCreate`) and no longer passed as `lmode` arguments to `CeedOperatorSetField` and `CeedElemRestrictionApply`.

4.0.4 Examples

libCEED-0.6 contains greatly expanded examples with new documentation. Notable additions include:

- Standalone `ex2-surface` (`examples/ceed/ex2-surface`): compute the area of a domain in 1, 2, and 3 dimensions by applying a Laplacian.
- PETSc `example-petsc-area` (`examples/petsc/area.c`): computes surface area of domains (like the cube and sphere) by direct integration on a surface mesh; demonstrates geometric dimension different from topological dimension.
- PETSc `example-petsc-bps`:
  - `examples/petsc/bpsraw.c` (formerly `bps.c`): transparent CUDA support.
  - `examples/petsc/bps.c` (formerly `bpsdmplex.c`): performance improvements and transparent CUDA support.
  - `example-petsc-bps-sphere` (`examples/petsc/bpssphere.c`): generalizations of all CEED BPs to the surface of the sphere; demonstrates geometric dimension different from topological dimension.
- `example-petsc-multigrid` (`examples/petsc/multigrid.c`): new p-multigrid solver with algebraic multigrid coarse solve.
- `example-petsc-navier-stokes` (`examples/fluids/navierstokes.c`; formerly `examples/navier-stokes`): unstructured grid support (using PETSc's `DMPlex`), implicit time integration, SU/SUPG stabilization, free-slip boundary conditions, and quasi-2D computational domain support.
- `example-petsc-elasticity` (`examples/solids/elasticity.c`): new solver for linear elasticity, small-strain hyperelasticity, and globalized finite-strain hyperelasticity using p-multigrid with algebraic multigrid coarse solve.

In what follows, we provide a detailed description of the added examples.

4.1 PETSc Area problems

This example is located in the subdirectory `examples/petsc`. It demonstrates a simple usage of libCEED with PETSc to calculate the surface area of a closed surface. The code uses higher level communication protocols for mesh handling in PETSc's DMPlex. This example has the same mathematical formulation as in the standalone example `examples/ceed/ex1-volume`, with the exception that the physical coordinates for this problem are $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$, while the coordinates of the reference element are $\boldsymbol{X} = (X, Y) \equiv (X_1, X_2) \in \mathbf{I} = [-1, 1]^2$.

4.1.1 Cube

This is one of the test cases of the computation of the surface area of a 2D manifold embedded in 3D. This problem can be run with::

```
./area -problem cube
```

This example uses the following coordinate transformations for the computation of the geometric factors: from the physical coordinates on the cube, denoted by $\bar{\mathbf{x}} = (\bar{x}, \bar{y}, \bar{z})$, and physical coordinates on the discrete surface, denoted by $\mathbf{x} = (x, y)$, to $\mathbf{X} = (X, Y) \in \mathbf{I} = [-1, 1]^2$ on the reference element, via the chain rule

$$\frac{\partial \mathbf{x}}{\partial \mathbf{X}}_{(2\times 2)} = \frac{\partial \mathbf{x}}{\partial \bar{\mathbf{x}}}_{(2\times 3)} \frac{\partial \bar{\mathbf{x}}}{\partial \mathbf{X}}_{(3\times 2)}, \tag{1}$$

with Jacobian determinant given by

$$|J| = \left\| col_1\left(\frac{\partial \bar{\mathbf{x}}}{\partial \mathbf{X}}\right)\right\| \left\| col_2\left(\frac{\partial \bar{\mathbf{x}}}{\partial \mathbf{X}}\right)\right\|. \tag{2}$$

We note that in equation (1), the right-most Jacobian matrix $\partial\bar{\mathbf{x}}/\partial\mathbf{X}_{(3\times2)}$ is provided by the library, while $\partial\mathbf{x}/\partial\bar{\mathbf{x}}_{(2\times3)}$ is provided by the user as

$$\left[col_1\left(\frac{\partial\bar{\mathbf{x}}}{\partial\mathbf{X}}\right)/\left\|col_1\left(\frac{\partial\bar{\mathbf{x}}}{\partial\mathbf{X}}\right)\right\|, col_2\left(\frac{\partial\bar{\mathbf{x}}}{\partial\mathbf{X}}\right)/\left\|col_2\left(\frac{\partial\bar{\mathbf{x}}}{\partial\mathbf{X}}\right)\right\|\right]^T_{(2\times3)}.$$

4.1.2 Sphere

This problem computes the surface area of a tensor-product discrete sphere, obtained by projecting a cube inscribed in a sphere onto the surface of the sphere (this discrete surface is sometimes referred to as a cubed-sphere). This problem can be run with::

```
./area -problem sphere
```

This example uses the following coordinate transformations for the computation of the geometric factors: from the physical coordinates on the sphere, denoted by $\mathring{\mathbf{x}} = (\mathring{x}, \mathring{y}, \mathring{z})$, and physical coordinates on the discrete surface, denoted by $\mathbf{x} = (x, y, z)$, to $\mathbf{X} = (X, Y) \in \mathbf{I} = [-1, 1]^2$ on the reference element, via the chain rule

$$\frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{X}_{(3\times2)}} = \frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{x}_{(3\times3)}}\frac{\partial\mathbf{x}}{\partial\mathbf{X}_{(3\times2)}}, \tag{3}$$

with Jacobian determinant given by

$$|J| = \left|col_1\left(\frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{X}}\right) \times col_2\left(\frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{X}}\right)\right|. \tag{4}$$

We note that in equation (3), the right-most Jacobian matrix $\partial\mathbf{x}/\partial\mathbf{X}_{(3\times2)}$ is provided by the library, while $\partial\mathring{\mathbf{x}}/\partial\mathbf{x}_{(3\times3)}$ is provided by the user with analytical derivatives. In particular, for a sphere of radius 1, we have

$$\mathring{\mathbf{x}}(\mathbf{x}) = \frac{1}{\|\mathbf{x}\|}\mathbf{x}_{(3\times1)}$$

and thus

$$\frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{x}} = \frac{1}{\|\mathbf{x}\|}\mathbf{I}_{(3\times3)} - \frac{1}{\|\mathbf{x}\|^3}(\mathbf{x}\mathbf{x}^T)_{(3\times3)}.$$

4.2 PETSc BPs on the cubed-sphere

For the L^2 projection problems, BP1-BP2, that use the mass operator, the coordinate transformations and the corresponding Jacobian determinant, equation (4), are the same as in the example in sec. 4.1.2. For the Poisson's problem, BP3-BP6, on the cubed-sphere, in addition to equation (4), the pseudo-inverse of $\partial\mathring{\mathbf{x}}/\partial\mathbf{X}$ is used to derive the contravariant metric tensor. We begin by expressing the Moore-Penrose (left) pseudo-inverse:

$$\frac{\partial\mathbf{X}}{\partial\mathring{\mathbf{x}}_{(2\times3)}} \equiv \left(\frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{X}}\right)^+_{(2\times3)} = \left(\frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{X}}^T_{(2\times3)}\frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{X}}_{(3\times2)}\right)^{-1}\frac{\partial\mathring{\mathbf{x}}}{\partial\mathbf{X}}^T_{(2\times3)}. \tag{5}$$

This enables computation of gradients of an arbitrary function $u(\mathring{\mathbf{x}})$ in the embedding space as

$$\frac{\partial u}{\partial\mathring{\mathbf{x}}_{(1\times3)}} = \frac{\partial u}{\partial\mathbf{X}_{(1\times2)}}\frac{\partial\mathbf{X}}{\partial\mathring{\mathbf{x}}_{(2\times3)}}$$

and thus the weak Laplacian may be expressed as

$$\int_\Omega \frac{\partial v}{\partial \overset{\circ}{\mathbf{x}}} \left(\frac{\partial u}{\partial \overset{\circ}{\mathbf{x}}}\right)^T = \int_\Omega \frac{\partial v}{\partial \mathbf{X}} \underbrace{\frac{\partial \mathbf{X}}{\partial \overset{\circ}{\mathbf{x}}} \left(\frac{\partial \mathbf{X}}{\partial \overset{\circ}{\mathbf{x}}}\right)^T}_{\mathbf{g}_{(2\times 2)}} \left(\frac{\partial u}{\partial \mathbf{X}}\right)^T \tag{6}$$

where we have identified the 2×2 contravariant metric tensor $\mathbf{g}$ (sometimes written $\mathbf{g}^{ij}$). This expression can be simplified to avoid the explicit Moore-Penrose pseudo-inverse,

$$\mathbf{g} = \left(\frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}\right)^{-1}_{(2\times 2)} \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T_{(2\times 3)} \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}_{(3\times 2)} \left(\frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}\right)^{-T}_{(2\times 2)} = \left(\frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}\right)^{-1}_{(2\times 2)} \tag{7}$$

where we have dropped the transpose due to symmetry. This allows us to simplify (6) as

$$\int_\Omega \frac{\partial v}{\partial \overset{\circ}{\mathbf{x}}} \left(\frac{\partial u}{\partial \overset{\circ}{\mathbf{x}}}\right)^T = \int_\Omega \frac{\partial v}{\partial \mathbf{X}} \underbrace{\left(\frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}\right)^{-1}}_{\mathbf{g}_{(2\times 2)}} \left(\frac{\partial u}{\partial \mathbf{X}}\right)^T,$$

which is the form implemented in `qfunctions/bps/bp3sphere.h`.

4.3 Solid mechanics elasticity mini-app

This example is located in the subdirectory `examples/solids`. It solves the steady-state static momentum balance equations using unstructured high-order finite/spectral element spatial discretizations. As for the `examples/fluids` case, the solid mechanics elasticity example has been developed using PETSc, so that the pointwise physics (defined at quadrature points) is separated from the parallelization and meshing concerns.

In this mini-app, we consider three formulations used in solid mechanics applications: linear elasticity, Neo-Hookean hyperelasticity at small strain, and Neo-Hookean hyperelasticity at finite strain. We provide the strong and weak forms of static balance of linear momentum in the small strain and finite strain regimes. The stress-strain relationship (constitutive law) for each of the material models is provided. Due to the nonlinearity of material models in Neo-Hookean hyperelasticity, the Newton linearization of the material models is provided.

Note: Linear elasticity and small-strain hyperelasticity can both by obtained from the finite-strain hyperelastic formulation by linearization of geometric and constitutive nonlinearities. The effect of these linearizations is sketched in the diagram below, where $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ are stress and strain, respectively, in the small strain regime, while $\boldsymbol{S}$ and $\boldsymbol{E}$ are their finite-strain generalizations (second Piola-Kirchoff tensor and Green-Lagrange strain tensor, respectively) defined in the reference configuration, and C is a linearized constitutive model.

$$\begin{array}{ccc} \text{Finite Strain Hyperelastic} & \overset{\text{constitutive}}{\longrightarrow}_{\boldsymbol{S}=\mathsf{C}\boldsymbol{E}} & \text{St. Venant-Kirchoff} \\ {\scriptstyle\text{geometric}}\ \downarrow\ \boldsymbol{E}\to\boldsymbol{\epsilon} & & {\scriptstyle\text{geometric}}\ \downarrow\ \boldsymbol{E}\to\boldsymbol{\epsilon} \\ \text{Small Strain Hyperelastic} & \overset{\text{constitutive}}{\longrightarrow}_{\boldsymbol{\sigma}=\mathsf{C}\boldsymbol{\epsilon}} & \text{Linear Elastic} \end{array} \tag{8}$$

4.3.1 Running the mini-app

The elasticity min-app is controlled via command-line options, the following of which are mandatory.

- `-mesh [filename]`: Path to mesh file in any format supported by PETSc.
- `-degree [int]`: Polynomial degree of the finite element basis

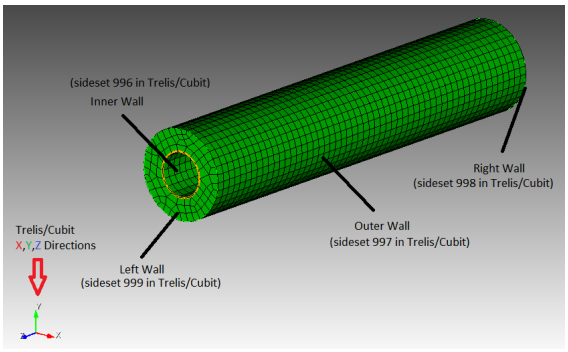

Figure 12: Example of mesh for sample run.

- `-E [real]`: Young's modulus, $E > 0$
- `-nu [real]`: Poisson's ratio, $\nu < 0.5$
- `-bc_zero [int list]`: List of faces sets on which to enforce zero displacement
- `-bc_clamp [int list]`: List of face sets on which to displace by `-bc_clamp_max` in the y direction

(One can set only one of `-bc_zero` or `-bc_clamp`, but the result will likely not be interesting.)

Note: This solver can use any mesh format that PETSc's `DMPlex` can read (Exodus, Gmsh, Med, etc.). Our tests have primarily been using Exodus meshes created using CUBIT; sample meshes used for the example runs suggested here can be found in https://github.com/jeremylt/ceedSampleMeshesthis repository. Note that many mesh formats require PETSc to be configured appropriately; e.g., `--download-exodusii` for Exodus support.

Consider the specific example of the mesh seen below:

With the sidesets defined in the figure, we provide here an example of a minimal set of command line options:

```
./elasticity -mesh [.exo file] -degree 4 -E 1e6 -nu 0.3 -bc_zero 999 -bc_clamp 998
```

In this example, we set the left boundary, face set 999, to zero displacement and the right boundary, face set 998, to displace by the default value of -1.0 in the y direction.

These command line options are the minimum requirements for the mini-app, but additional options may also be set.

To verify the convergence of the linear elasticity formulation on a given mesh with the method of manufactured solutions, run:

```
./elasticity -mesh [mesh] -degree [degree] -nu [nu] -E [E] -forcing mms
```

This option attempts to recover a known solution from an analytically computed forcing term.

4.3.2 On algebraic solvers

This mini-app is configured to use the following Newton-Krylov-Multigrid method by default.

- Newton-type methods for the nonlinear solve, with the hyperelasticity models globalized using load increments.
- Preconditioned conjugate gradients to solve the symmetric positive definite linear systems arising at each Newton step.
- Preconditioning via p-version multigrid coarsening to linear elements, with algebraic multigrid (PETSc's `GAMG`) for the coarse solve. The default smoother uses degree 3 Chebyshev with Jacobi preconditioning. (Lower degree is often faster, albeit less robust; try `-outer_mg_levels_ksp_max_it 2`, for example.)

Application of the linear operators for all levels with degree $p > 1$ is performed matrix-free using analytic Newton linearization, while the lowest order $p = 1$ operators are assembled explicitly (using coloring at present).

Many related solvers can be implemented by composing PETSc command-line options.

4.3.3 Nondimensionalization

Quantities such as the Young's modulus vary over many orders of magnitude, and thus can lead to poorly scaled equations. One can nondimensionalize the model by choosing an alternate system of units, such that displacements and residuals are of reasonable scales.

- `-units_meter`: 1 meter in scaled length units
- `-units_second`: 1 second in scaled time units
- `-units_kilogram`: 1 kilogram in scaled mass units

For example, consider a problem involving metals subject to gravity.

- Displacement: $\boldsymbol{u}$, 1cm $= 10^{-2}$ m
- Young's modulus: E, 100 GPa $= 10^{11}\,\mathrm{kg\,m^{-1}\,s^{-2}}$
- Body force (gravity) on volume: $\int \rho \boldsymbol{g}$, $5 \cdot 10^4\,\mathrm{kg\,m^{-2}\,s^{-2}} \cdot (\texttt{volume}\,\mathrm{m}^3)$

One can choose units of displacement independently (e.g., `-units_meter 100` to measure displacement in centimeters), but E and $\int \rho \boldsymbol{g}$ have the same dependence on mass and time, so cannot both be made of order 1. This reflects the fact that both quantities are not equally significant for a given displacement size; the relative significance of gravity increases as the domain size grows.

4.4 Linear Elasticity

The strong form of the static balance of linear momentum at small strain for the three-dimensional linear elasticity problem is given by [14]:

$$\nabla \cdot \boldsymbol{\sigma} + \boldsymbol{g} = \boldsymbol{0} \tag{9}$$

where $\boldsymbol{\sigma}$ and $\boldsymbol{g}$ are stress and forcing functions, respectively. We multiply (9) by a test function $\boldsymbol{v}$ and integrate the divergence term by parts to arrive at the weak form: find $\boldsymbol{u} \in \mathcal{V} \subset H^1(\Omega)$ such that

$$\int_\Omega \nabla \boldsymbol{v} \colon \boldsymbol{\sigma}\, dV - \int_{\partial\Omega} \boldsymbol{v} \cdot (\boldsymbol{\sigma} \cdot \hat{\boldsymbol{n}})\, dS - \int_\Omega \boldsymbol{v} \cdot \boldsymbol{g}\, dV = 0, \quad \forall \boldsymbol{v} \in \mathcal{V}, \tag{10}$$

where $\boldsymbol{\sigma} \cdot \hat{\boldsymbol{n}}|_{\partial\Omega}$ is replaced by an applied force/traction boundary condition.
The constitutive law (stress-strain relationship) is given by:

$$\boldsymbol{\sigma} = \mathsf{C} \colon \boldsymbol{\epsilon}, \tag{11}$$

where

$$\boldsymbol{\epsilon} = \frac{1}{2}\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right) \tag{12}$$

is the symmetric (small/infinitesimal) strain tensor and the colon represents a double contraction (over both indices of $\boldsymbol{\epsilon}$). For notational convenience, we express the symmetric second order tensors $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ as vectors

of length 6 using the Voigt notation. Hence, the fourth order elasticity tensor C (also known as elastic moduli tensor or material stiffness tensor) can be represented as a 6×6 symmetric matrix

$$\mathsf{C} = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & & & \\ \nu & 1-\nu & \nu & & & \\ \nu & \nu & 1-\nu & & & \\ & & & \frac{1-2\nu}{2} & & \\ & & & & \frac{1-2\nu}{2} & \\ & & & & & \frac{1-2\nu}{2} \end{pmatrix}, \tag{13}$$

where E is the Young's modulus and ν is the Poisson's ratio.

4.4.1 Lamé parameters

An alternative formulation, in terms of the Lamé parameters,

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{14}$$

$$\mu = \frac{E}{2(1+\nu)} \tag{15}$$

can be found. In this formulation, the constitutive equation (11) may be written as

$$\boldsymbol{\sigma} = \lambda(\operatorname{trace}\boldsymbol{\epsilon})\boldsymbol{I}_3 + 2\mu\boldsymbol{\epsilon},$$

where $\boldsymbol{I}_3$ is the 3×3 identity matrix. With the formulation using Lamé parameters, the elasticity tensor (13) becomes

$$\mathsf{C} = \begin{pmatrix} \lambda+2\mu & \lambda & \lambda & & & \\ \lambda & \lambda+2\mu & \lambda & & & \\ \lambda & \lambda & \lambda+2\mu & & & \\ & & & \mu & & \\ & & & & \mu & \\ & & & & & \mu \end{pmatrix}. \tag{16}$$

Note that the incompressible limit $\nu \to \frac{1}{2}$ causes $\lambda \to \infty$, and thus C becomes singular.

4.5 Hyperelasticity at Small Strain

The strong and weak forms given above, in (9) and (10), are valid for Neo-Hookean hyperelasticity at small strain. However, the constitutive law differs and is given as follows:

$$\boldsymbol{\sigma} = \lambda \log(1 + \operatorname{trace}\boldsymbol{\epsilon})\boldsymbol{I}_3 + 2\mu\boldsymbol{\epsilon} \tag{17}$$

where $\boldsymbol{\epsilon}$ is defined as in (12).

4.5.1 Newton linearization

Due to nonlinearity in the constitutive law, we require a Newton linearization of (17). To derive the Newton linearization, we begin by expressing the derivative,

$$d\boldsymbol{\sigma} = \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\epsilon}} : d\boldsymbol{\epsilon}$$

where

$$d\boldsymbol{\epsilon} = \frac{1}{2}\left(\nabla d\boldsymbol{u} + \nabla d\boldsymbol{u}^T\right)$$

and

$$d\nabla\boldsymbol{u} = \nabla d\boldsymbol{u}.$$

Therefore,

$$d\boldsymbol{\sigma} = \bar{\lambda} \cdot \operatorname{trace} d\boldsymbol{\epsilon} \cdot \boldsymbol{I}_3 + 2\mu d\boldsymbol{\epsilon} \tag{18}$$

where we have introduced the symbol

$$\bar{\lambda} = \frac{\lambda}{1 + \boldsymbol{\epsilon}_v}$$

where volumetric strain is given by $\boldsymbol{\epsilon}_v = \sum_i \boldsymbol{\epsilon}_{ii}$.
Equation (18) can be written in Voigt matrix notation as follows:

$$\begin{pmatrix} d\sigma_{11} \\ d\sigma_{22} \\ d\sigma_{33} \\ d\sigma_{23} \\ d\sigma_{13} \\ d\sigma_{12} \end{pmatrix} = \begin{pmatrix} 2\mu+\bar{\lambda} & \bar{\lambda} & \bar{\lambda} & & & \\ \bar{\lambda} & 2\mu+\bar{\lambda} & \bar{\lambda} & & & \\ \bar{\lambda} & \bar{\lambda} & 2\mu+\bar{\lambda} & & & \\ & & & \mu & & \\ & & & & \mu & \\ & & & & & \mu \end{pmatrix} \begin{pmatrix} d\epsilon_{11} \\ d\epsilon_{22} \\ d\epsilon_{33} \\ 2d\epsilon_{23} \\ 2d\epsilon_{13} \\ 2d\epsilon_{12} \end{pmatrix}. \tag{19}$$

4.6 Hyperelasticity at Finite Strain

In the *total Lagrangian* approach for the Neo-Hookean hyperelasticity problem, the discrete equations are formulated with respect to the reference configuration. In this formulation, we solve for displacement $\boldsymbol{u}(\boldsymbol{X})$ in the reference frame $\boldsymbol{X}$. The notation for elasticity at finite strain is inspired by [13] to distinguish between the current and reference configurations. We denote by capital letters the reference frame and by small letters the current one.

The strong form of the static balance of linear-momentum at *finite strain* (total Lagrangian) is given by:

$$-\nabla_X \cdot \boldsymbol{P} - \rho_0 \boldsymbol{g} = \boldsymbol{0} \tag{20}$$

where the $_X$ in ∇_X indicates that the gradient is calculated with respect to the reference configuration in the finite strain regime. $\boldsymbol{P}$ and $\boldsymbol{g}$ are the *first Piola-Kirchhoff stress* tensor and the prescribed forcing function, respectively. ρ_0 is known as the *reference* mass density. The tensor $\boldsymbol{P}$ is not symmetric, living in the current configuration on the left and the reference configuration on the right.

$\boldsymbol{P}$ can be decomposed as

$$\boldsymbol{P} = \boldsymbol{F}\,\boldsymbol{S}, \tag{21}$$

where $\boldsymbol{S}$ is the *second Piola-Kirchhoff stress* tensor, a symmetric tensor defined entirely in the reference configuration, and $\boldsymbol{F} = \boldsymbol{I}_3 + \nabla_X \boldsymbol{u}$ is the deformation gradient. Different constitutive models can define $\boldsymbol{S}$.

4.6.1 Constitutive modeling

In their most general form, constitutive models define $\boldsymbol{S}$ in terms of state variables. In the model taken into consideration in the present mini-app, the state variables are constituted by the vector displacement field $\boldsymbol{u}$, and its gradient $\nabla_X \boldsymbol{u}$. We begin by defining two symmetric tensors in the reference configuration, the right Cauchy-Green tensor

$$\boldsymbol{C} = \boldsymbol{F}^T \boldsymbol{F}$$

and the Green-Lagrange strain tensor

$$\boldsymbol{E} = \frac{1}{2}(\boldsymbol{C} - \boldsymbol{I}_3) = \frac{1}{2}\Big(\nabla_X \boldsymbol{u} + (\nabla_X \boldsymbol{u})^T + (\nabla_X \boldsymbol{u})^T \nabla_X \boldsymbol{u}\Big), \tag{22}$$

the latter of which converges to the linear strain tensor $\boldsymbol{\epsilon}$ in the small-deformation limit. The constitutive models considered, appropriate for large deformations, express $\boldsymbol{S}$ as a function of $\boldsymbol{E}$, similar to the linear case, shown in equation (11), which expresses the relationship between $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$. This constitutive model $\boldsymbol{S}(\boldsymbol{E})$ is a nonlinear tensor-valued function of a tensor-valued input, but an arbitrary choice of such a function will generally not be invariant under orthogonal transformations and thus will not admissible as a physical model must not depend on the coordinate system chosen to express it. In particular, given an orthogonal transformation Q, we desire

$$Q\boldsymbol{S}(\boldsymbol{E})Q^T = \boldsymbol{S}(Q\boldsymbol{E}Q^T), \tag{23}$$

which means that we can change our reference frame before or after computing $\boldsymbol{S}$, and get the same result either way. Constitutive relations in which $\boldsymbol{S}$ is uniquely determined by $\boldsymbol{E}$ (equivalently, $\boldsymbol{C}$ or related tensors) while satisfying the invariance property (23) are known as Cauchy elastic materials. Here, we focus on an important subset of them known as hyperelastic materials, for which we may define a strain energy density functional $\Phi(\boldsymbol{E}) \in \mathbb{R}$ and obtain the strain energy from its gradient,

$$\boldsymbol{S}(\boldsymbol{E}) = \frac{\partial \Phi}{\partial \boldsymbol{E}}. \tag{24}$$

Note: The strain energy density functional cannot be an arbitrary function $\Phi(\boldsymbol{E})$; it can only depend on *invariants*, scalar-valued functions γ satisfying

$$\gamma(\boldsymbol{E}) = \gamma(Q\boldsymbol{E}Q^T)$$

for all orthogonal matrices Q.

Consequently, we may assume without loss of generality that $\boldsymbol{E}$ is diagonal and take its set of eigenvalues as the invariants. It is clear that there can be only three invariants, and there are many alternate choices, such as $\operatorname{trace}(\boldsymbol{E}), \operatorname{trace}(\boldsymbol{E}^2), |\boldsymbol{E}|$, and combinations thereof. It is common in the literature for invariants to be taken from $\boldsymbol{C} = \boldsymbol{I}_3 + 2\boldsymbol{E}$ instead of $\boldsymbol{E}$.

For example, if we take the compressible Neo-Hookean model,

$$\Phi(\boldsymbol{E}) = \frac{\lambda}{2}(\log J)^2 + \frac{\mu}{2}(\operatorname{trace} \boldsymbol{C} - 3) - \mu \log J \tag{25}$$

$$= \frac{\lambda}{2}(\log J)^2 + \mu \operatorname{trace} \boldsymbol{E} - \mu \log J, \tag{26}$$

where $J = |\boldsymbol{F}| = \sqrt{|\boldsymbol{C}|}$ is the determinant of deformation (i.e., volume change) and λ and μ are the Lamé parameters in the infinitesimal strain limit.

To evaluate (24), we make use of

$$\frac{\partial J}{\partial \boldsymbol{E}} = \frac{\partial \sqrt{|\boldsymbol{C}|}}{\partial \boldsymbol{E}} = |\boldsymbol{C}|^{-1/2} |\boldsymbol{C}| \boldsymbol{C}^{-1} = J\boldsymbol{C}^{-1},$$

where the factor of $\frac{1}{2}$ has been absorbed due to $\boldsymbol{C} = \boldsymbol{I}_3 + 2\boldsymbol{E}$. Carrying through the differentiation (24) for the model (25), we arrive at

$$\boldsymbol{S} = \lambda \log J \boldsymbol{C}^{-1} + \mu(\boldsymbol{I}_3 - \boldsymbol{C}^{-1}). \tag{27}$$

Tip: An equivalent form of 27 is

$$\boldsymbol{S} = \lambda \log J \boldsymbol{C}^{-1} + 2\mu \boldsymbol{C}^{-1}\boldsymbol{E},$$

which is more numerically stable for small $\boldsymbol{E}$, and thus preferred for computation. Note that the product $\boldsymbol{C}^{-1}\boldsymbol{E}$ is also symmetric, and that $\boldsymbol{E}$ should be computed using (22).

Similarly, it is preferable to compute $\log J$ using `log1p`, especially in case of nearly incompressible materials. To sketch this idea, suppose we have the 2×2 symmetric matrix $C = \left(\begin{smallmatrix} 1+e_{00} & e_{01} \\ e_{01} & 1+e_{11} \end{smallmatrix}\right)$. Then we compute

$$\log \sqrt{|C|} = \frac{1}{2}\texttt{log1p}(e_{00} + e_{11} + e_{00}e_{11} - e_{01}^2).$$

which gives accurate results even in the limit when the entries e_{ij} are very small. For example, if $e_{ij} \sim 10^{-8}$, then naive computation of $\boldsymbol{I}_3 - \boldsymbol{C}^{-1}$ and $\log J$ will have a relative accuracy of order 10^{-8} in double precision and no correct digits in single precision. When using the stable choices above, these quantities retain full $\varepsilon_{\text{machine}}$ relative accuracy.

Note: One can linearize (27) around $\boldsymbol{E} = 0$, for which $\boldsymbol{C} = \boldsymbol{I}_3 + 2\boldsymbol{E} \to \boldsymbol{I}_3$ and $J \to 1 + \operatorname{trace} \boldsymbol{E}$, therefore (27) reduces to

$$\boldsymbol{S} = \lambda(\operatorname{trace} \boldsymbol{E})\boldsymbol{I}_3 + 2\mu\boldsymbol{E}, \tag{28}$$

which is the St. Venant-Kirchoff model.

This model can be used for geometrically nonlinear mechanics (e.g., snap-through of thin structures), but is inappropriate for large strain.

Alternatively, one can drop geometric nonlinearities, $\boldsymbol{E} \to \boldsymbol{\epsilon}$ and $\boldsymbol{C} \to \boldsymbol{I}_3$, while retaining the nonlinear dependence on $J \to 1 + \operatorname{trace} \boldsymbol{\epsilon}$, thereby yielding (17).

4.6.2 Weak form

We multiply (20) by a test function $\boldsymbol{v}$ and integrate by parts to obtain the weak form for finite-strain hyperelasticity: find $\boldsymbol{u} \in \mathcal{V} \subset H^1(\Omega_0)$ such that

$$\int_{\Omega_0} \nabla_X \boldsymbol{v} : \boldsymbol{P}\, dV - \int_{\Omega_0} \boldsymbol{v} \cdot \rho_0 \boldsymbol{g}\, dV - \int_{\partial\Omega_0} \boldsymbol{v} \cdot (\boldsymbol{P} \cdot \hat{\boldsymbol{N}})\, dS = 0, \quad \forall \boldsymbol{v} \in \mathcal{V}, \tag{29}$$

where $\boldsymbol{P} \cdot \hat{\boldsymbol{N}}|_{\partial\Omega}$ is replaced by any prescribed force/traction boundary conditions written in terms of the reference configuration. This equation contains material/constitutive nonlinearities in defining $\boldsymbol{S}(\boldsymbol{E})$, as well as geometric nonlinearities through $\boldsymbol{P} = \boldsymbol{F}\,\boldsymbol{S}$, $\boldsymbol{E}(\boldsymbol{F})$, and the body force $\boldsymbol{g}$, which must be pulled back from the current configuration to the reference configuration. Discretization of (29) produces a finite-dimensional system of nonlinear algebraic equations, which we solve using Newton-Raphson methods. One attractive feature of Galerkin discretization is that we can arrive at the same linear system by discretizing the Newton linearization of the continuous form; that is, discretization and differentiation (Newton linearization) commute.

4.6.3 Newton linearization

To derive a Newton linearization of (29), we begin by expressing the derivative of (21) in incremental form,

$$d\boldsymbol{P} = \frac{\partial \boldsymbol{P}}{\partial \boldsymbol{F}} : d\boldsymbol{F} = d\boldsymbol{F}\,\boldsymbol{S} + \boldsymbol{F}\,\underbrace{\frac{\partial \boldsymbol{S}}{\partial \boldsymbol{E}} : d\boldsymbol{E}}_{d\boldsymbol{S}} \tag{30}$$

where

$$d\boldsymbol{E} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{F}} : d\boldsymbol{F} = \frac{1}{2}\Big(d\boldsymbol{F}^T\boldsymbol{F} + \boldsymbol{F}^T d\boldsymbol{F}\Big).$$

The quantity $\frac{\partial \boldsymbol{S}}{\partial \boldsymbol{E}}$ is known as the incremental elasticity tensor, and is analogous to the linear elasticity tensor C of (13). We now evaluate $d\boldsymbol{S}$ for the Neo-Hookean model (27),

$$d\boldsymbol{S} = \frac{\partial \boldsymbol{S}}{\partial \boldsymbol{E}} : d\boldsymbol{E} = \lambda(\boldsymbol{C}^{-1} : d\boldsymbol{E})\boldsymbol{C}^{-1} + 2(\mu - \lambda \log J)\boldsymbol{C}^{-1} d\boldsymbol{E}\, \boldsymbol{C}^{-1}, \tag{31}$$

where we have used

$$d\boldsymbol{C}^{-1} = \frac{\partial \boldsymbol{C}^{-1}}{\partial \boldsymbol{E}} : d\boldsymbol{E} = -2\boldsymbol{C}^{-1} d\boldsymbol{E}\, \boldsymbol{C}^{-1}.$$

Note: In the small-strain limit, $\boldsymbol{C} \to \boldsymbol{I}_3$ and $\log J \to 0$, thereby reducing (31) to the St. Venant-Kirchoff model (28).

Note: Some cancellation is possible (at the expense of symmetry) if we substitute (31) into (30),

$$\begin{aligned} d\boldsymbol{P} &= d\boldsymbol{F}\, \boldsymbol{S} + \lambda(\boldsymbol{C}^{-1} : d\boldsymbol{E})\boldsymbol{F}^{-T} + 2(\mu - \lambda \log J)\boldsymbol{F}^{-T} d\boldsymbol{E}\, \boldsymbol{C}^{-1} && (32)\\ &= d\boldsymbol{F}\, \boldsymbol{S} + \lambda(\boldsymbol{F}^{-T} : d\boldsymbol{F})\boldsymbol{F}^{-T} + (\mu - \lambda \log J)\boldsymbol{F}^{-T}(\boldsymbol{F}^T d\boldsymbol{F} + d\boldsymbol{F}^T \boldsymbol{F})\boldsymbol{C}^{-1} && (33)\\ &= d\boldsymbol{F}\, \boldsymbol{S} + \lambda(\boldsymbol{F}^{-T} : d\boldsymbol{F})\boldsymbol{F}^{-T} + (\mu - \lambda \log J)\Big(d\boldsymbol{F}\, \boldsymbol{C}^{-1} + \boldsymbol{F}^{-T} d\boldsymbol{F}^T \boldsymbol{F}^{-T}\Big), && (34) \end{aligned}$$

where we have exploited $\boldsymbol{F}\boldsymbol{C}^{-1} = \boldsymbol{F}^{-T}$ and

$$\begin{aligned} \boldsymbol{C}^{-1} : d\boldsymbol{E} = \boldsymbol{C}^{-1}_{IJ} d\boldsymbol{E}_{IJ} &= \frac{1}{2}\boldsymbol{F}^{-1}_{Ik}\boldsymbol{F}^{-1}_{Jk}(\boldsymbol{F}_{\ell I} d\boldsymbol{F}_{\ell J} + d\boldsymbol{F}_{\ell I}\boldsymbol{F}_{\ell J}) && (35)\\ &= \frac{1}{2}\Big(\delta_{\ell k}\boldsymbol{F}^{-1}_{Jk} d\boldsymbol{F}_{\ell J} + \delta_{\ell k}\boldsymbol{F}^{-1}_{Ik} d\boldsymbol{F}_{\ell I}\Big) && (36)\\ &= \boldsymbol{F}^{-1}_{Ik} d\boldsymbol{F}_{kI} = \boldsymbol{F}^{-T} : d\boldsymbol{F}. && (37) \end{aligned}$$

We prefer to compute with (31) because (32) is more expensive, requiring access to (non-symmetric) $\boldsymbol{F}^{-1}$ in addition to (symmetric) $\boldsymbol{C}^{-1} = \boldsymbol{F}^{-1}\boldsymbol{F}^{-T}$, having fewer symmetries to exploit in contractions, and being less numerically stable.

It is sometimes useful to express (31) in index notation,

$$\begin{aligned} d\boldsymbol{S}_{IJ} &= \frac{\partial \boldsymbol{S}_{IJ}}{\partial \boldsymbol{E}_{KL}} d\boldsymbol{E}_{KL} && (38)\\ &= \lambda(\boldsymbol{C}^{-1}_{KL} d\boldsymbol{E}_{KL})\boldsymbol{C}^{-1}_{IJ} + 2(\mu - \lambda \log J)\boldsymbol{C}^{-1}_{IK} d\boldsymbol{E}_{KL}\boldsymbol{C}^{-1}_{LJ} && (39)\\ &= \underbrace{\Big(\lambda \boldsymbol{C}^{-1}_{IJ}\boldsymbol{C}^{-1}_{KL} + 2(\mu - \lambda \log J)\boldsymbol{C}^{-1}_{IK}\boldsymbol{C}^{-1}_{JL}\Big)}_{\mathsf{C}_{IJKL}} d\boldsymbol{E}_{KL}\,, && (40) \end{aligned}$$

where we have identified the effective elasticity tensor $\mathsf{C} = \mathsf{C}_{IJKL}$. It is generally not desirable to store C, but rather to use the earlier expressions so that only 3×3 tensors (most of which are symmetric) must be manipulated. That is, given the linearization point $\boldsymbol{F}$ and solution increment $d\boldsymbol{F} = \nabla_X(d\boldsymbol{u})$ (which we are solving for in the Newton step), we compute $d\boldsymbol{P}$ via

1. recover $\boldsymbol{C}^{-1}$ and $\log J$ (either stored at quadrature points or recomputed),
2. proceed with 3×3 matrix products as in (31) or the second line of (38) to compute $d\boldsymbol{S}$ while avoiding computation or storage of higher order tensors, and

3. conclude by (30), where $\boldsymbol{S}$ is either stored or recomputed from its definition exactly as in the nonlinear residual evaluation.

Note: The decision of whether to recompute or store functions of the current state $\boldsymbol{F}$ depend on a roofline analysis [17, 5] of the computation and the cost of the constitutive model. For low-order elements where flops tend to be in surplus relative to memory bandwidth, recomputation is likely to be preferable, where as the opposite may be true for high-order elements. Similarly, analysis with a simple constitutive model may see better performance while storing little or nothing while an expensive model such as Arruda-Boyce [4], which contains many special functions, may be faster when using more storage to avoid recomputation. In the case where complete linearization is preferred, note the symmetry $\mathsf{C}_{IJKL} = \mathsf{C}_{KLIJ}$ evident in (38), thus C can be stored as a symmetric 6×6 matrix, which has 21 unique entries. Along with 6 entries for $\boldsymbol{S}$, this totals 27 entries of overhead compared to computing everything from $\boldsymbol{F}$. This compares with 13 entries of overhead for direct storage of $\{\boldsymbol{S}, \boldsymbol{C}^{-1}, \log J\}$, which is sufficient for the Neo-Hookean model to avoid all but matrix products.

5. CEED-3.0 RELEASE

The CEED 3.0 release contains a suite of the following 12 integrated packages, which range from low-level modular libraries to applications, plus the CEED meta-package.

GSLIB-1.0.6

Laghos-3.0

libCEED-0.6 New website, numerous new features, backend performance performance, and examples/mini-apps; see section 4 for details.

MAGMA-2.5.3

MFEM-4.1

Nek5000-19.0

Nekbone-17.0

NekCEM-c8db04b

OCCA-1.0.9

PETSc-3.13 Enhanced GPU support, scalability and capability improvements for distributed unstructured meshes and finite element spaces, and much more; see release notes for details.

PUMI-2.2.2

Remhos-1.0

The CEED suite is primarily distributed using the Spack package manager, and can be installed on any system using `spack install ceed`. For convenience, we provide configurations for common systems including Mac OSX, Linux (RHEL7 and Ubuntu), LLNL Lassen, and ORNL Summit. We also provide base and complete pre-built images for use with Docker, Singularity, and Shifter, enabling users to deploy CEED technologies in seconds, and to incorporate into continuous integration and cloud environments. See the distribution website for further details.

6. APPLICATIONS GPU/CPU PERFORMANCE AND CAPABILITIES IMPROVEMENTS

CEED aims to impact a wide range of ECP applications through close collaborations, by developing easy-to-use discretization libraries and high-order finite-element algorithms for critical needs in the applications. Our primary goal of this subtask is to help the CEED-engaged applications (ExaSMR, MARBL, ExaWind, ExaAM) with their GPU porting, performance improvements and the development of new capabilities that require high-order research and development (meshing, solvers, physics models, etc.). As part of this subtask, we are reaching out to additional applications in the ECP and also in non-ECP projects to support CEED's high-order technologies.

CEED-T4 Sub-tasks. (see ECP Jira: Items marked with ✓are considered to be completed or equivalent outcomes have been achieved.

- ✓ Help improve performance/capabilities of ExaSMR (see Section 6.1).
- ✓ Help improve performance/capabilities of MARBL (see Section 6.2).
- ✓ Help improve performance/capabilities of ExaWind (see Section 6.3).
- ✓ Help improve performance/capabilities of ExaAM (see Section 6.4).
- ✓ Reach out to additional applications, that interested in CEED's high-order technologies, such as E3SM, NRC, VTO (see Section 6.5).
- ✓ Methods for the automatic generation of hex and/or mixed meshes as needed by the CEED applications will be considered (see Section 6.1).

Completion Criteria. The following completion criteria are described and documented within this report.

- ✓ Documented performance and capabilities improvements in the ECP/CEED applications to be included in the CEED-MS34 (ADCD04-53) report.

6.1 ExaSMR

CEED team has developed an all-hex meshing utility for efficient simulation of flow through random arrays of dense-packed spheres. This capability is central to analysis of newly designed pebble bed reactors. The utility generates all-hex meshes by tessellating highly-optimized Voronoi cells surrounding each pebble. The new approach yields low element counts (a factor of six lower than previous approaches based on tet-to-hex conversion), which allows the use of more accurate, higher-order, elements for the same number of grid points. The improved mesh quality also results in lower iteration counts and less severe stability constraints, such that the overall runtime is reduced by an order of magnitude compared to the tet-to-hex approach. Development work was started on 146 pebbles and recently extended to a 500,000 element mesh for a 1568-pebble simulation. Improved performance on GPUs and CPUs using the latest repo versions of NekRS and Nek5000 during the period of March 2020 are demonstrated in this section.

6.1.1 Performance on GPUs and CPUs for Pebble Beds Reactor Simulations

Figure 13 demonstrates NekRS simulation capability running for a 1568-pebble all-hex mesh that has been run on 66 GPUs on Summit using a total number of grids $n = 179,864,398$ with $E = 524,386$ and $N = 7$. It shows turbulent flow profile around the dense-packed 1568 pebbles in a cylinder for Reynolds number of 10,000. Figure 14 validates the similar behaviors of GMRES iterations in pressure solve for different number of GPUs and examine the accumulated walltime per timestep. Using 6 GPU per node, the performance is measured on 11, 22, and 44 nodes on Summit (i.e., 66, 132, and 264 GPUs, respectively) that corresponds to the number of grid points per GPU (n/gpu) $2.7M$, $1.3M$, and $680K$. Table 2 shows strong-scaling performance with the accumulated walltime at 5000 timestep demonstrating improved parallel efficiency on GPU using relatively lower count of grid points per GPU, compared to our previous baseline studies. We achieve 75% parallel efficiency using 1.3M points per GPU and 52% using $680K$ points per GPU.

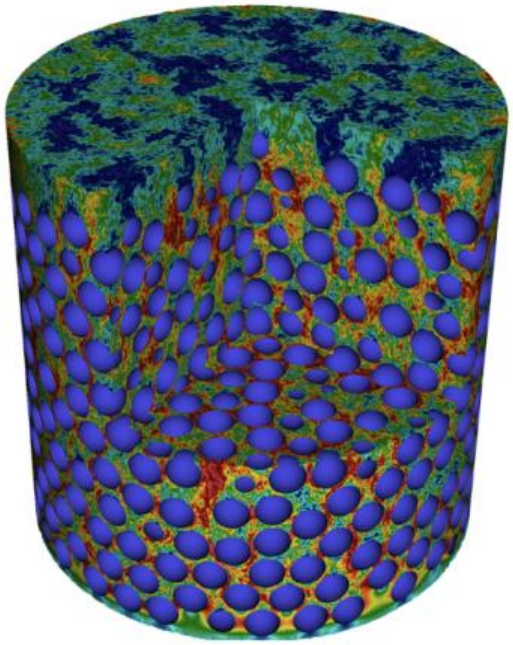

Figure 13: NekRS simulation for 1568 pebbles using an all-hex mesh with $E = 524,386$ and $N = 7$ (total grids $n = 179,864,398$) on Summit's 66 GPUs, demonstrating velocity profile for $Re = 10,000$.

nodes	GPUs	E/GPU	n/GPU	1000 steps	5000 steps	ratio	efficiency
11	66	7945	2,725,135	7.70385e+02 (sec)	5.22546e+3 (sec)	–	1.0
22	132	3972	1,362,396	5.05101e+02 (sec)	3.45225e+3 (sec)	1.5	0.75
44	264	1986	681,198	3.62689e+02 (sec)	2.49199e+3 (sec)	2.1	0.52

Table 2: NekRS Strong-scaling performance on Summit GPUs, measured at 1000 and 5000 timesteps for 1568 pebbles in Figure 13 with $E = 365844$.

Figure 15 demonstrates Nek5000 performance comparison between CEED–developed all-hex and ExaSMR–developed tet-to-hex meshes for a geometry consisting of 146 pebbles. We consider performance for a similar resolution. All-hex mesh has total grids of $n = 21M$ with ($E = 63132$, $N = 7$) and tet-to-hex mesh has total grids of $n = 23M$ with ($E = 365844$, $N = 4$). We examine the GMRES iteration behaviors of pressure solve and simulation time for different timestep sizes, $dt = 1e-3, 2e-3, 3e-3$. We observe significant redution in the pressure iterations when using all-hex mesh (20 iterations in average) and 3x reduction in time-to-solution for the best performing case of all-hex and test-to-hex with $dt = 3e-3$. Next section describes more detail of our all-hex meshing strategies.

6.1.2 Novel All-Hex Meshing Strategies for Dense-Packed Spheres for Pebble Beds

Modeling flow through randomly-packed spherical beds is a common scenario in science and engineering applications. For high-Reynolds number flows, $Re_{D_h} \gg 1$, where D_h is the hydraulic diameter of the void space, high-order methods, which have minimal numerical dissipation and dispersion, are highly effective for tracking turbulent structures in the flow. The spectral element method (SEM)[6], which uses local tensor-product bases on curvilinear brick elements, is particularly efficient with memory costs scaling as $O(n)$, independent of local approximation order N. Here, $n \approx EN^3$ is the total number of gridpoints in the mesh comprising E elements. In contrast, standard p-type finite element methods, which support tetrahedral elements, exhibit $O(EN^6)$ storage costs. Altenernative tet-supporting high-order formulations (Dubrinov)

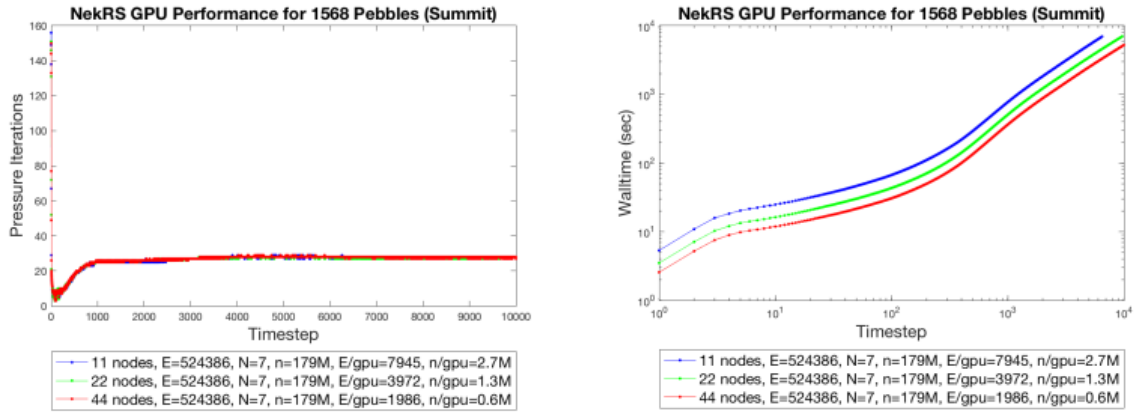

Figure 14: NekRS GPU performance on Summit's 11, 22, and 44 nodes using total 66, 132, and 264 GPUs, respetively, for the 1568-pebbles mesh in Figure 13. GMRES interations and accumulated walltime measured per timestep are shown.

have a 6-fold increase in cost over the standard SEM formulation.

For a given resolution, n, the use of high-order elements with $N = 7$–15 implies a 300- to 3000-fold reduction in the number of elements required when compared to linear elements. The meshing task is thus somewhat more challenging as the objective is to have a high quality mesh with relatively few elements. In contrast, with linear tets or hexes, one has the opportunity to effectively fill the computational domain with grains of sand and repair connections where needed. Paving is one all-hex example that exhibits this point. For the dense-packed sphere problem, the distance between the boundaries and the center of the voids is not large–paved surfaces will quickly collide and a large number of (smaller) elements will be required to resolve many of the configurations.

An alternative all-hex strategy is to first discretize the void space with tets and to then convert each tet to four hexes. Given the efficiency of the SEM, this idea is not as terrible as it may seem. For one thing, all-tet meshes tend to yield fairly isotropic elements that yield favorable iteration counts (c.f., [10]). This strategy has been pursued in a recent article by Yuan *et al.* [18]. Unfortunately, the tet-to-hex strategy leads to relatively high element counts for the dense-packed sphere problem. Yuan and coworkers found that they could only use $N = 3$ for the target Reynolds numbers in their spherical beds, which is suboptimal for the SEM where $N > 5$ is preferred [7]. For example, the tet-to-hex strategy of Yuan *et al.* yielded a mesh with E =375000 elements for 146 spheres in a cylindrical container.

CEED team developed novel meshing strategies for generating high-quality hexahedral element meshs that ensure accurate representation of densely packed spheres for very complex pebble-bed reactor geometries. Our target is to capture highly turbulent flow structures in solution at minimal cost, by using minimum resolution and better accuracy with high-order approximation. The algorithmic strategies discussed in details include efficient edge collapse, tessellation, smoothing, and projection. We demonstrate various pebble bed geometries ranging from hundreds to thousands of pebbles with quality measurements, provided with flows simulations, validation and performance. The underlying discretizations are based on spectral element method[6] and simulations are performed using Nek5000 [8]. The algorithmic strategy builds stone-steps toward simulating millions of pebbles for the full-core reactor at exascale.

While the interstitial space in randomly packed spheres is quite complex, there are several feature of this problem that make it possible to recast the meshing question into a sequence of simpler, local, problems, making the overall problem far more tractable. First, the presence of so many surfaces provides a large number of termination points for a given mesh topology. As anyone who has expended significant effort on meshing knows, such surfaces are a welcome relief from the nightmare of having to merge multiple incoming mesh topologies. Second, by decomposing the domain into Voronoi cells defined by the sphere centers, we can localize the meshing problem in several ways.

First, we reduce the problem to that of building a mesh that fills the gap between the facets of the Voronoi

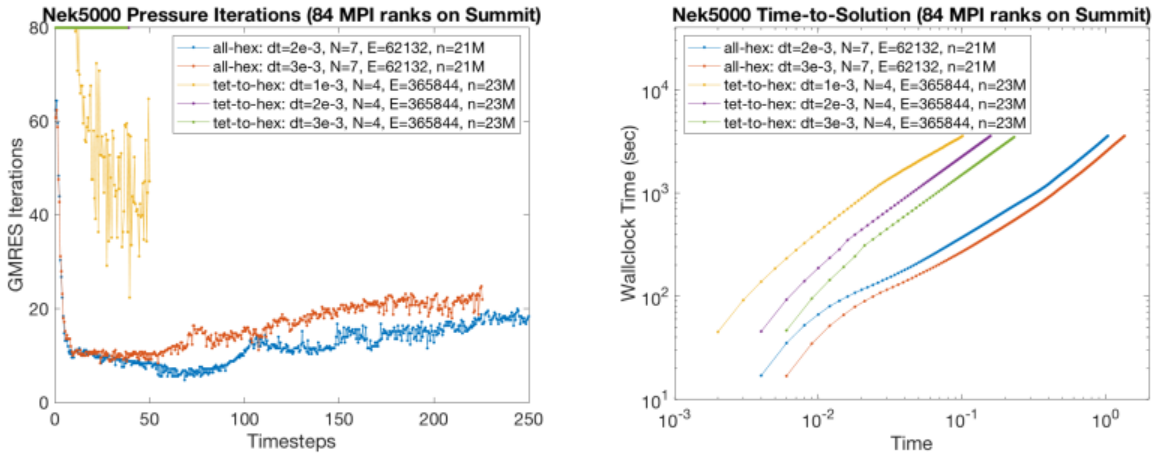

Figure 15: GMRES iterations (left) and time-to-solution (right) comparison between all-hex and tet-to-hex meshes. Meshes represent 146 pebbles for a pebble-beds reactor geometry.

cell and the sphere surface. This process entails tesselating each Voronoi facet into an all-quad decomposition and projecting these quads onto the sphere surface. The convexity of the Voronoi cell and co-planarity of the facets ensures that this decomposition is possible. Where desired, refinement in the radial direction is always possible without disturbing other cells.

Second, each tesselation of each facet is a local problem. Because of bilateral symmetry about the Voronoi facet, tesselation of a facet that is valid for one sphere will be valid also for the sphere on the opposite side of the facet. We note that facets may have an odd number of vertices. If, for example, the facet is a triangle, then the all-quad tesselation will require using midside subdivision, resulting in the introction of midside nodes along each edge. To retain the locality of the meshing strategy, we therefore introduce midside nodes on each edge of each facet. With an even number of vertices thus guaranteed, we can generate an all-quad tesselation of the resulting polygon. The strategy outlined above forms the essence of the proposed algorithm. In principle, it will produce a base mesh with relatively few elements that inherit reasonable shape qualities from the Voronoi base. There are several important modifications to put the method into practice. We mention these briefly as: short-edge collapse (to remove small facets); corner replacement (to improve void-center resolution and overall mesh quality); touching-sphere tesselation (to avoid contact singularity); mesh refinement; mesh smoothing/optimization (to improve final mesh quality); surface projection (to ensure that the final SEM nodal points are on the sphere surfaces while avoiding mesh entanglement). Save for the last step, which is implemented as a user-defined functionality in the spectral element code Nek5000, all of the other parts of the algorithm are implemented in matlab in $O(N_s)$ or $O(N_s log N_s)$ time, where N_s is the number of spheres.

6.2 MARBL

In this section we summarize the recent GPU-related work in MARBL, including the improvements in the Laghos miniapp. We start by introducing the newly developed Remhos (REMap High-Order Solver) miniapp, which is modeled after the DG remap phase of MARBL.

6.2.1 Initial release of the Remhos miniapp

The CEED team has completed the initial version of the Remhos miniapp. Remhos solves the pure advection equations that are used to perform discontinuous field interpolation (remap) as part of the Eulerian phase in Arbitrary-Lagrangian Eulerian (ALE) simulations. The goal of the miniapp to obtain high-order accurate, conservative, and bounds-preserving solution of the advection equation. The miniapp implements some of the newest ideas for this problem in the context of high-order Discontinuous Galerkin (DG) finite elements,

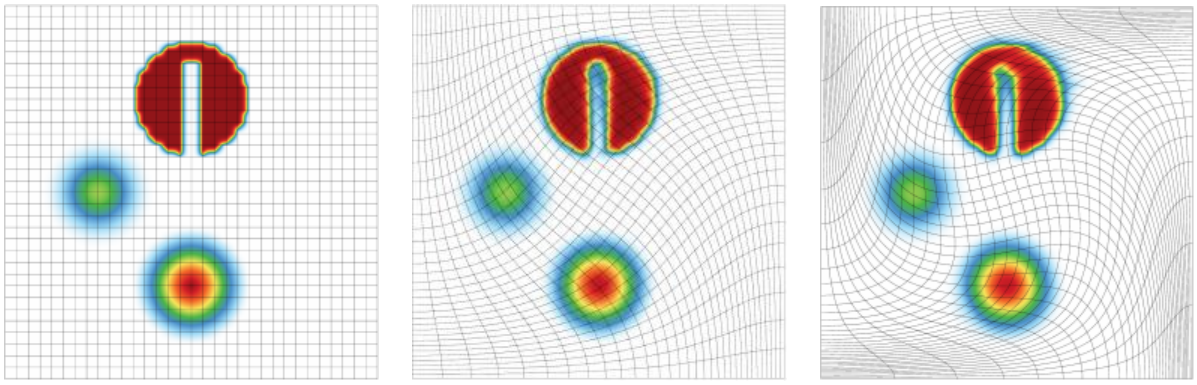

Figure 16: Example from Remhos demonstrating a remap calculation with different amounts of mesh distortion.

namely, the discretization methods described in the articles [2, 1, 3, 11, 12].

The problem Remhos is solving is formulated as a time-dependent system of ordinary differential equations (ODEs) for the unknown coefficients of a high-order finite element (FE) function. The left-hand side of this system is controlled by a DG mass matrix, while the right-hand side is constructed from a DG advection matrix. This miniapp supports the full assembly and partial assembly options for deriving and solving the ODE system. As usual, partial assembly is the main algorithm of interest for high orders. For low orders (e.g. 2nd order in 3D), optimizing both algorithms is of interest.

Remhos supports two execution modes, namely, *transport* and *remap*, which result in slightly different algebraic operators. The main difference between the two modes is that in the case of remap, the mass and advection matrices change in time, while they are constant for the transport case.

Other computational motives in Remhos include the following:

- Support for unstructured meshes, in 2D and 3D, with quadrilateral and hexahedral elements. Serial and parallel mesh refinement options can be set via a command-line flag.
- Explicit time-stepping loop with a variety of time integrator options. Remhos supports Runge-Kutta ODE solvers of orders 1, 2, 3, 4 and 6.
- Discontinuous high-order finite element discretization spaces of runtime-specified order.
- Moving (high-order) meshes.
- Mass operator that is local per each zone. It is inverted by iterative or exact methods at each time step. This operator is constant in time (transport mode) or changing in time (remap mode). Options for full or partial assembly.
- Advection operator that couples neighboring zones. It is applied once at each time step. This operator is constant in time (transport mode) or changing in time (remap mode). Options for full or partial assembly.
- Domain-decomposed MPI parallelism.
- Optional in-situ visualization with GLVis (http:/glvis.org) and data output for visualization and data analysis with VisIt (http://visit.llnl.gov).

An example of a remap test in shown in Figure 16. The release of Remhos-1.0 is planned for the end of March 2020.

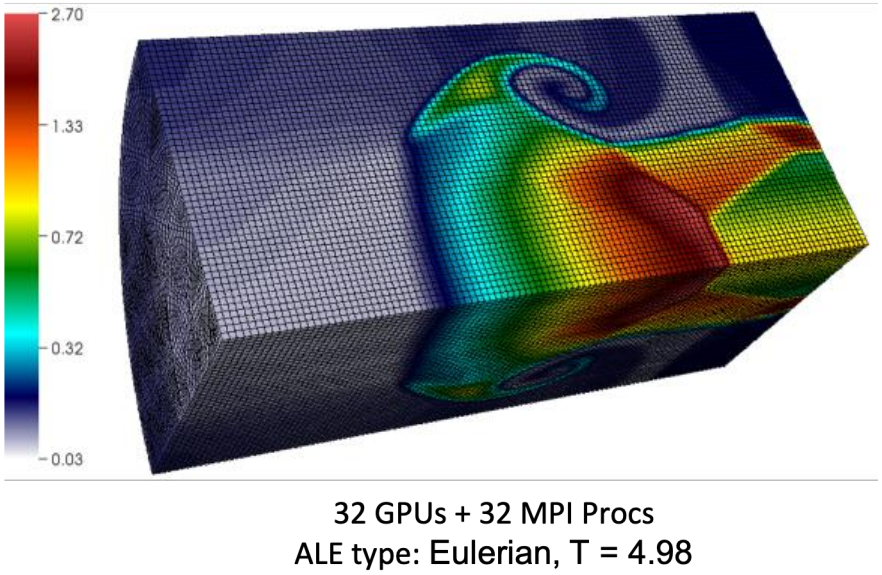

Figure 17: MARBL result obtained on 32 NVIDIA Tesla V100 GPUs at LLNL.

6.2.2 Laghos miniapp improvements

By utilizing the new capabilities of MFEM-4.1, the performance and codebase of Laghos are improved significantly. The MFEM-4.1 code infrastructure allows to combine all major versions of Laghos (baseline CPU, CUDA, HIP, OCCA and RAJA, as explained in MS-25) into a single reusable source code. This eliminates the need of having different source codes into different folders for each version, allowing reusability, easier maintenance, and direct access to future optimizations.

More importantly, the original partial assembly kernels are replaced by the connection to the optimized MFEM routines, e.g., methods for mass action, calculation of geometric factors, evaluations of FE functions at quadrature points. Laghos also utilizes the MFEM's newly developed GPU-based small dense matrix and vector operations, which are used by its quadrature-level physics-related computations.

The above improvements will be available in the next release, Laghos-3.0, which is planned for the end of March 2020.

6.2.3 GPU work in MARBL

The focus of the GPU efforts in MARBL has been porting the Lagrangian phase of the simulation, where the mesh moves with the material velocity, and the remap phase, where solution fields are transferred from one mesh to another. The MARBL team has been utilizing the GPU infrastructure of MFEM-4.1 and collaborating with the CEED researchers to port and optimize various sections of the code. In the period November 2019 - March 2020, the MARBL team has been able to reduce the computational time of the GPU-enabled tests by a factor of around 4x. Preliminary results have demonstrated performance improvements between 2x and 10x in select routines when comparing a node of Power 9 + V100 (4 MPI ranks, 1 MPI rank per GPU) to Intel Xeon E5-2695 (36 MPI ranks). Since not all parts of the code have been ported to the GPU, large memory transfers are still needed to complete a full simulation. These memory transfers are still the main bottleneck. Result obtained by a GPU simulation is shown in Figure 17. In this section we give more details about the latest GPU additions.

Lagrangian phase GPU work Since this phase is what the Laghos miniapp models, the work in MARBL has been aided by the GPU implementation available in Laghos. This phase has three major components: inversion of a global CG mass matrix, computation of physics-specific quadrature point data, and application of a force operator.

The global CG mass matrix inversion is fully ported on GPUs. The device mass action kernels are very similar to the ones in MFEM, but currently live in the BLAST code, as the previous MFEM implementation did not support non-constant integral coefficients. Since such coefficients has been recently implemented in MFEM-4.1, the MARBL team is planning to switch to using the optimized MFEM versions.

The calculation of quadrature point data, i.e., forming the D matrices for the mass and the force operators, has similar structure as Laghos, but requires many additional physics-specific computations. These methods make heavy use of element-local small vector and matrix operations, e.g., for computation of geometric factors and quadrature interpolation. For these, the MARBL team is utilizing the recently introduced GPU implementations in MFEM-4.1. This is an ongoing effort and not all parts of the source have been ported.

The action of the force operator has been fully ported. The MARBL implementation utilizes the partial assembly kernels from Laghos, while the GPU-specific routines have been implemented locally. The MARBL team is planning to replace these with the implementation that has recently become available in Laghos.

Unlike Laghos, the Lagrangian phase of MARBL contains a computation of a hyperviscosity coefficient, which involves consecutive applications of a Laplacian operator. This method has also been ported on the GPU by applying directly the MFEM's optimized diffusion integrator kernels.

Remap phase GPU work The remap algorithm has two main components, namely, velocity remap, which is a CG advection solve, and remap of other fields, which is modeled by DG advection. The DG method is nonlinear, involving 3 separate components, namely, a high-order (HO) method, a low-order (LO) method and a nonlinear flux-corrected transport (FCT) procedure. Currently, all of these three components require sparse matrices.

The CG advection solve is fully ported, which includes the computation of quadrature data and application of the CG mass and advection matrices. Similarly to the CG mass matrix inversion in the Lagrangian phase, the remap GPU code is implemented inside MARBL, and switching to using the optimized MFEM kernels is planned.

Using the MFEM infrastructure, the MARBL developers have developed custom GPU code to populate the sparsity of the advection sparse matrix, thus achieving LO and HO implementations for the GPU. It is expected that this approach will be improved significantly by the work in Remhos, as it contains matrix-free methods to obtain LO and HO solutions. The FCT procedure is also fully ported GPU.

Future GPU work As already mentioned, there are ongoing efforts to transition from customized GPU kernels to their optimized MFEM versions, e.g., for the application of the force operator and the CG mass matrices. The MARBL team will also continue to integrate the MFEM's new GPU routines for small dense matrices and vectors. It is expected that using the MFEM kernels will lead to better performance. The CEED developers will keep developing the Laghos and Remhos miniapps, both in terms of GPU performance and matrix-free methods capabilities, and these new methods and implementations would be readily available to be used in MARBL.

6.3 ExaWind

In collaboration with the ExaWind team, we have developed high-order atmospheric boundary layer (ABL) models with the ultimate goal of simulating wakes in the farfield regions of wind farms. These regions have minimal geometric complexity and would be well-suited to treatment by high-order discretizations that can deliver the required accuracy with an order-of-magnitude fewer gridpoints than second-order schemes [15].

This work involves efforts by Ananias Tomboulides (AUTH, ANL) from the CEED team and Mathew Churchfield, Shreyas Ananthan, Michael Sprague from the ExaWind team at NREL. The plan is to develop a bake-off study involving Nek5000/NekRS, Naluwind, and AMReX that will be reported as a journal article. Here we describe the background on ABL and provide current status of this ongoing effort with some preliminary performance results.

6.3.1 LES Simulations for Atmospheric Boundary Layer Model

Efficient simulation of atmospheric boundary layer (ABL) flows is important for the study of wind farms, urban canyons, and basic weather modeling. The ABL is directly affected by the Earth's surface. When

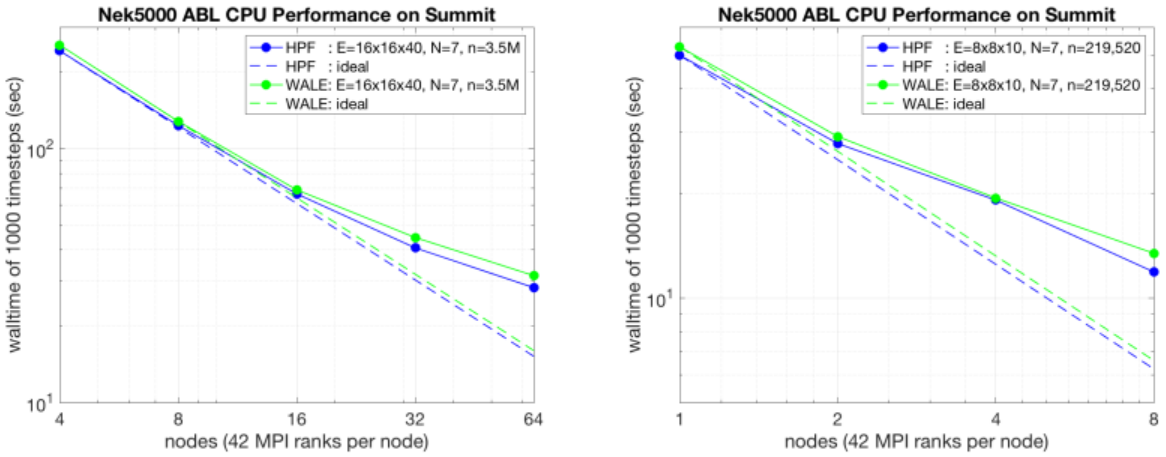

Figure 18: Nek5000 strong-scaling performance for ABL simulations ($Re = 50000$) using $E = 10240$ and $E = 640$ with $N = 7$ on Summit. (Left) Running on 672 cores (16 nodes, 42 cores per node) using 5226 points per core, it reaches strong-scale limit (90% efficiency) with averaged timing of 0.07 seconds per timestep. (Right) Running 84 cores (2 nodes, 42 cores per node) using 2613 points per core, it is below strong-scale limit with 56 % efficiency.

the surface is colder than the air, the layer close to the ground becomes a stably stratified boundary layer (SBL), which is classified according to the intensity of the thermal stratification. SBL flows can be some of the most damaging flows for wind turbines. These flows can typically be simulated by using the Boussinesq approximation which also allows propagation of gravity waves. The lower turbulence levels mean that turbine wakes persist for longer distances downstream, and hence can decrease the efficiency of a wind plant. The stable ABL can also contain a low-level jet, a layer of flow that has speed greater than the flow above the ABL.

All of these features of the stably stratified ABL make it important and challenging to simulate accurately. Moreover, changes in the large eddy simulation (LES) turbulence model used can lead to large differences in flow and heat transfer predictions. For unstable atmospheres, where deep natural convection can have a range of kilometers, the Boussinesq approximation is no longer valid and the anelastic model must be used. In the anelastic model, the pressure, potential temperature and density applied in the thermodynamics typically come from the environmental hydrostatically balanced pressure profile and the equations are free from acoustic wave time-stepping restrictions.

6.3.2 Test Problems and Baseline Performance on Summit

In collaboration with ExaWind researchers, the CEED team has focused on developing reliable high-fidelity LES-model implementations for stable ABL flows as well as development of improved wall models. Continuation of this effort will involve performance comparisons on CPUs and GPUs between three different codes, Nek5000/NekRS, NaluWind, and AMReX, representing unstructured or structured high-order, structured low-order, and block-structure AMR discretizations, respectively. Proposed configurations for the bake-off study are the following.

- Perform scaling studies and comparison on CPUs and GPUs.
- Use box geometries representing physical domain $[400m \times 400m \times 400m]$, or multiples thereof for weak-scaling studies.
- Implement traction boundary conditions (BCs) horizontally at the bottom; symmetry BCs on the top; and periodic BCs in the horizontal directions. The results of the traction BC have to be very close to those of highly-refined simulations having no-slip BCs at the bottom.

- Initiate tests with resolutions of a coarser mesh, $E = 640$ ($= 8 \times 8 \times 10$), and a finer mesh, $E = 10240$ ($= 16 \times 16 \times 20$), with $N = 7$, representing total grids $n = 219,520$ and $n = 3,512,320$, respectively.
- Use statistically same initial condition (restart or perturbed one) once the numerical solution evolves to a developed (but still transient) turbulent state and run a specific range of physical time units (not fixed number of timesteps).
- Study accuracy, convergence and performance of two different LES models, HPF (high-pass filter) and WALE models, are to be demonstrated for a fixed resolution.
- Use Reynolds numbers in the range of $Re = 50,000$ to $Re = 100,000$.

The CEED team was able to develop SEM algorithms and validate the implementation of the ABL model in Nek5000. Figure 18 demonstrates preliminary results of ABL simulations for Reynolds number of 50,000 on Summit's CPUs with strong-scaling studies for two different resolutions using $E = 10240$ and $E = 640$ with $N = 7$. Figure 18, left, shows 90% efficiency on 672 cores (16 nodes, 42 cores per node) using 5226 points per core, with averaged timing of 0.07 seconds per timestep – this is close to its strong-scale limit. Figure 18, right, shows 56 % efficiency on 84 cores (2 nodes, 42 cores per node) using 2613 points per core - this is below the strong-scale limit of Nek5000.

As the next step, the CEED team is planning to extend ABL implementation in Nek5000 (CPU) to the new NekRS (GPU) code. The ExaWind team is currently moving forward with the testing stage of their ABL implementation in NaluWind and AMReX. Weekly telecons between the CEED and Exawind teams have helped to solidify the problem definition and feasible approaches to robust implementations.

6.4 ExaAM

Fundamentally, the ExaAM activity must couple microstructure development and local property analysis with process simulation. Given the length scales, physical mechanisms, and response characteristics of interest, finite element codes employing crystal-mechanics-based constitutive models are the appropriate computational approach for the microstructure-to-properties aspect of the overall ExaAM problem. A survey of crystal-mechanics-based codes has indicated that none of the existing codes are suited to ExaAM needs – due to a combination of limitations on distribution and lack of suitability to exascale computing platforms. The ExaAM team is therefore creating an application specifically for local property analysis. This development is in collaboration with the CEED team. ExaConstit first release is focused on the ExaAM local property analysis, but as required, additional physics, inline results processing, and other features will be added to meet wider ExaAM needs. Significant integration between the ExaAM and CEED ECP activities is planned during this process. The application has been released under a BSD-3 clause license under the LLNL GitHub page [https://github.com/LLNL/ExaConstit].

6.4.1 ExaConstit miniapp

The main development work for this quarter has been transitioning ExaConstit over to a finite element formulation that can run on the GPU. This new formulation is based on partial assembly formulation of the linearized portion of the nonlinear PDE. Partial assembly is based on the idea of being able to take advantage of tensor contraction operations in-order to achieve the lowest number of FLOP count and memory storage requirements. The partial assembly formulation is primarily advantageous for higher order elements as described in [https://doi.org/10.1016/j.procs.2016.05.302]. Outside of this reformulation, ExaConstit was refactored to take advantage of the various parallelization abstraction layers introduced in MFEM v4.0 to allow our various compute and material kernels to run on the GPU.

Over the course of the ExaAM activity, we plan to make use of the MFEM capabilities for higher-order finite elements to achieve unprecedented fidelity in the resolution of microstructure-level material response features that are important for resolving initiation events such as the nucleation and growth of porosity.

6.4.2 Test Problem

The test problems from a recent ExaAM milestone considered here are centered around an ExaCA-generated [ExaCA/FY19Q1] microstructure for 316L [FY18Q3] stainless steel. One model being used is from ExaCMech

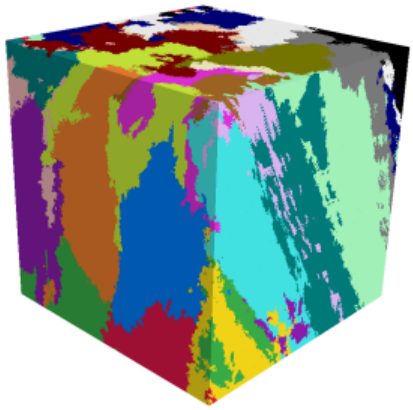

Figure 19: ExaCA generated grain microstructure that approximates AM processes for a 8e6 element mesh.

that make use of a Voce hardening model. The Voce hardening model is a basic crystal plasticity model but contains enough physics for our current needs. The mesh is $100 \times 100 \times 100 = 10^6$ elements, and this mesh is a discretization of a polycrystal with 500 grains. This problem size is chosen to be relevant to anticipated long-term challenge problem needs, in terms of the number of grains involved. As mentioned below, the challenge problem is likely to ultimately require finer discretization (more elements per grain) to accurately inform macro-scale models.

An additional ExaCA-generated microstructure produced from an analytical heat transfer solution to mimic an AM built part is examined as seen in the below figure. The mesh is $200 \times 200 \times 200 = 8 \times 10^6$ elements and is a discretization of a polycrystal with 380 grains.

Some development activities between ExaConsit and CEED:

- Refactored ExaConstit in order to be able to make use of a partial assembly formulation for the linear solve portion of the Newton Raphson solve
- Formulated a partial assembly formulation for general solid mechanics problems
- Implemented this formulation into ExaConstit
- Converted most major loops over to using `MFEM_FORALL` loops which allows the flexibility at runtime to choose between different backend to run the kernels on (CUDA, HIP, OpenMP, CPU, RAJA-CUDA, RAJA-OpenMP, RAJA-CPU, ... etc)
- Initial demonstration of running everything on the GPU has shown promising results compared to the old CPU full assembly performance

Results from Demonstration Problem. The main results for the ExaAM milestone were focused on porting the main compute kernels of ExaConstit over to running on the GPU. The relevant physics capabilities for the demonstration problem were already demonstrated in [ExaConstit/FY19Q4]. ExaConstit has been ported to Summit. ExaConstit and its dependent libraries have all been compiled using the gcc 7.3 and 8.1 compiler suites in combination with the NVCC compiler suite. They have also been compiled against the clang 6.0 compiler combined with the use of gfortran 7.3 for the UMATs.

Performance. A set of simulations were run using ExaConstit on the CPUs only and GPUs. These runs provided information about the strong scaling of ExaConstit while being run on CPUs and GPUs for ExaAM-relevant calculations. As described in platform status above, these simulations were run on Summit. The full ExaCA generated microstructure is used on a mesh that uses c3d8 type elements (with 8 quadrature points per finite element). The simulations were taken out to 1% strain in tension. The figure

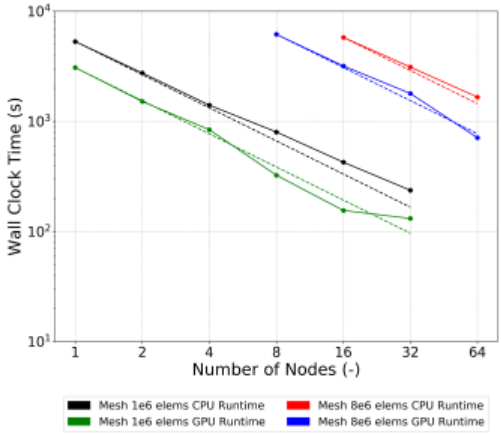

Figure 20: Strong scale study conducted on Summit ExaCMech model for CPU and GPUs.

Nodes (#)	CPU Time(s)	GPU Time(s)	GPU Scaling Factor (-)
1	5307	3078	1.72
2	2747	1513	1.82
4	1402	839	1.67
8	800	324	2.47
16	425	155	2.74
32	236	131	1.80

Table 3: Timings of strong scale study conducted on Summit for 1e6 element mesh.

Nodes (#)	CPU Time(s)	GPU Time(s)	GPU Scaling Factor (-)
8	-	6166	-
16	5785	3182	1.82
32	3110	1789	1.74
64	1659	709	2.34

Table 4: Timings of strong scale study conducted on Summit for 8e6 element mesh.

below summarizes the strong scaling as a function of time and number of nodes for both the CPU only and GPU runs. The wall clock time for each simulation is provided in a table down below to highlight the difference in performance in the two different runtimes along with the scaling factor of the GPU over the CPU. The dashed lines in the below highlight what runtimes perfect strong scaling would have. Any points below the dashed line represent super scaling and those above represent less than perfect scaling.

Based on the above, one of the most interesting observation is the appearance of a super-scaling phenomenon within the GPUs once a given workload per GPU is reached. If we plot this workload per GPU in Figure 3 it is apparent that this starts around 125k quadrature points per GPU. As the GPU code becomes more performant, it'll need to be seen if these same observations continue to exist.

Challenge Problem Implications. While automated sampling strategies for concurrent multi-scale modeling [http://dx.doi.org/10.1002/nme.3071] may ultimately be used in the challenge problem, in the near-term ExaConstit will be used in an offline mode to obtain parameters for macro-scale constitutive models to be used in the part-scale build simulations. In this offline strategy, a human will decide on the number of microstructural conditions at which ExaConstit simulations are used to probe microstructure-dependent

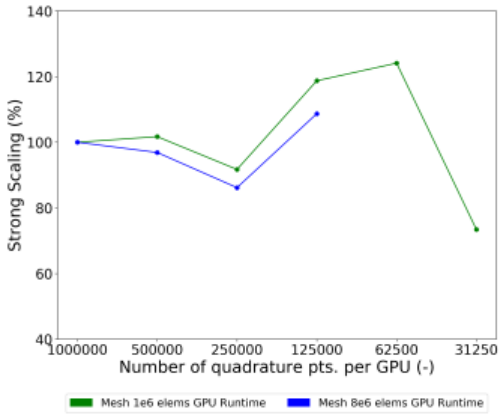

Figure 21: GPU strong scale study as a function of data residing on each GPU.

material response. With tens of calculations to probe each microstructure and each calculation involving up to 107 integration points or more, and tens of time steps per simulation; there is significant computational expense. However with a human-in-the-loop initial approach, we will be able to modulate the workload, with corresponding impact on computational fidelity, to keep the simulations tractable given currently available computational resources. As we gain more knowledge on computational expense and on the most effective means of informing macro-scale constitutive models, this coupling strategy will be refined.

It is worth noting that a grand challenge in advanced manufacturing is design for control of microstructure and attendant material response. In a design optimization setting there could be the need to run many iterations of the challenge problem, and in such a setting we would envision ExaConstit calculations populating a manifold in a feature space of microstructures (potentially 100 or more dimensional) [http://dx.doi.org/10.1016/j.cma.2014.09.017].

For the challenge problem, we will keep an eye on potential advances that are needed, say for capturing sub-scale microstructural heterogeneities such as precipitates or non-uniform solute segregation, and focused experiments are being considered that would more directly constrain the detailed predictions of ExaConstit simulation results. But for the near-term, needs for the challenge problem center around computational performance for exascale-relevant computing platforms.

Next Steps. In the near term, the main priorities of the project center around parameterizing the models for as-built IN625 materials through on-going experiments and determining the appropriate representative volume element (RVE) size for the challenge problem. The RVE study will be conducted in coordination with the ExaCA component of the project. Secondary priorities include the examination of different techniques to speed up convergence of the global finite element solution. This may include examination of matrix-free preconditioners and of finite element formulations.

6.5 Additional Applications: E3SM, NRC, VTO

The CEED team is reaching out to additional applications in the ECP and also to non-ECP projects from funded by DOE and other agencies (e.g., the DOE's Vehicle Technology Office; the Nuclear Regulatory Commission) to leverage CEED's high-order technologies.

- **E3SM.** In collaboration with E3SM, the CEED team is identifying a set of Helmholtz problems on the surface of a sphere as a relevant benchmark problem. The Helmholtz equations are derived from the primitive equations describing atmospheric flow and represent the challenging fast gravity waves that otherwise restrict the timestep size in the general circulation model if treated explicitly. The plan is to develop efficient algorithms for solving 128 Helmholtz equations, where each of the equations represents the pressure level at a different vertical wavenumber. The dynamical core for E3SM is based on a spectral element method in the horizontal directions, which is directly aligned with CEED's high-order motif. Preconditioning strategies for these high-order methods are essential to make the proposed

semi-implicit strategy competitive with pure explicit timestepping. Several groups in CEED and E3SM have expressed interest in holding a bake-off to identify the fastest strategy. Once the study identifies a fast scheme with low iteration counts, the hope to guide E3SM in developing a fast semi-implicit timestepping approach.

- **Nuclear Regulatory Commission (NRC).** Nek5000 and NekRS have been designated to help the DOE and the Nuclear Regulatory Commission in licensing processes for nuclear reactor technology. Through the new initiative, members of the CEED project will work with the DOE's National Reactor Innovation Center to provide the NRC with thermal-hydraulics modeling codes. DOE will also provide the NRC access to state-of-the-art computing capabilities in support licensing of advanced reactors. Access to these updated codes and facilities will help expedite the review process and be used to predict expected reactor operations to reduce the time it takes to validate and certify new designs, enabling a faster commercialization process.
- **Vehicle Technology Office (VTO).** Argonne scientists in the Energy Systems Divison have been funded by DOE's Vehicle Technology Office to use Nek5000 for multicycle simulations of internal combustion engines. Argonne CEED team members are indirectly supporting this project through assistance with modeling, meshing, and code performance. The GPU simulation capabilities of NekRS will be a critical component for success of this project.

7. OTHER PROJECT ACTIVITIES

7.1 BP paper

One characteristic common to many science and engineering applications in high-performance computing (HPC) is their enormous range of spatial and temporal scales, which can manifest as billions of degrees of freedom (DOFs) to be updated over tens of thousands of timesteps. Typically, the large number of spatial scales is addressed through parallelism—processing all elements of the spatial problem simultaneously—while temporal complexity is most efficiently addressed sequentially, particularly in the case of highly nonlinear problems such as fluid flow, which defy linear transformations that might expose additional parallelism (e.g., through frequency-domain analysis). Simulation campaigns for these problems can require weeks or months of wall-clock time on the world's fastest supercomputers. One of the principal objectives of HPC is to reduce these runtimes to manageable levels.

We explored performance and space-time trade-offs for computational model problems that typify the important compute-intensive kernels of large-scale numerical solvers for partial differential equations (PDEs) that govern a wide range of physical applications. We are interested in peak performance (degrees of freedom per second) on a fixed number of nodes and in minimum time to solution at reasonable parallel efficiencies, as would be experienced by computational scientists in practice. While we consider matrix-free implementations of p-type finite and spectral element methods as the principal vehicle for our study, the performance analysis presented here is relevant to a broad spectrum of numerical PDE solvers, including finite difference, finite volume, and h-type finite elements, and thus is widely applicable.

Performance tests and analyses are critical to effective HPC software development and are central components of CEED project. One of the foundational components of CEED is a sequence of PDE-motivated *bake-off* problems designed to establish best practices for efficient high-order methods across a variety of platforms. The idea is to pool the efforts of several high-order development groups to identify effective code optimization strategies for each architecture. Our first round of tests features comparisons from the software development projects Nek5000, MFEM, deal.II, and libParaNumal.

This effort has been reported as a paper, *"Scalability of high-performance PDE solvers"* by P. Fischer, M. Min, T. Rathnayake, S. Dutta, T. Kolev, V. Dobrev, J.-S. Camier, M. Kronbichler, T. Warburton, K. Swirydowicz, and J. Brown, and the paper has been accepted for publication in the International Journal of High Performance Computing Applications [9].

7.2 CEED Third Annual Meeting

The third annual meeting of the CEED co-design center took place August 6–8 at Virginia Tech, Blacksburg, VA. Participants reported on the progress in the center, deepened existing and established new connections with ECP hardware vendors, ECP software technologies projects and other collaborators, planned project activities and brainstormed / worked 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.

The meeting included 53 participants from 18 institutions and 15 projects. Further details on the meeting are available through the CEED webpage: `https://ceed.exascaleproject.org/ceed3am/`.

7.3 Initial NekRS release

NekRS (`https://github.com/Nek5000/nekRS`) is a newly developed C++ version of Nek5000, built on top of libParanumal to support highly optimized kernels on GPUs using OCCA (`https://libocca.org`). NekRS has been used for large scale simulations, currently focusing on ExaSMR problems with pebble beds and 17×17 rod geometries, and also ported to Aurora development system to get baseline performance on Intel Gen9, compared to Nvidia V100.

7.4 Aurora Workshop at ANL

The CEED team participated the Aurora Programming (Sept. 17–19, 2019) held at ANL. The team successfully ported NekRS on the Aurora development system (Intel Gen9) during the workshop, and was able to get the baseline performance in comparison to NVIDIA V100 as shown in Table 1. Turbulent pipe flow is simulated for 100 time steps for the problem size of $E = 9234$ and $N = 7$.

7.5 Outreach

CEED researchers were involved in a number of outreach activities, including collaborating with NVIDIA to optimize the Laghos miniapp for Summit/Sierra, release of the libParanumal bake-off kernels in a standalone benchmarking suite, benchParanumal, preparing a breakout session on high-order methods and applications at the ECP annual meeting, minisymposium proposal for the International Conference in Spectral and High-Order Methods (ICOSAHOM20, 14 papers (“Scalability of high-performance PDE solvers”, “Nonconforming Mesh Refinement for High-Order Finite Elements”, “Fast Batched Matrix Multiplication for Small Sizes using Half-Precision Arithmetic on GPUs”, “Towards Simulation-Driven Optimization of High-Order Meshes by the Target-Matrix Optimization Paradigm”, “Preparation and Optimization of a Diverse Workload for a Large-Scale Heterogeneous System”, “ClangJIT: Enhancing C++ with Just-in-Time Compilation”, “On the use of LES-based turbulent thermal-stress models for rod bundle simulations”, “OpenACC acceleration for the Pn-Pn-2 algorithm in Nek5000”, “Nonlinear artificial viscosity for spectral element methods”, “Nonconforming Schwarz-spectral element methods for incompressible flow”, “Scalable low-order finite element preconditioners for high-order spectral element Poisson solvers”, “A Characteristic-Based Spectral Element Method for Moving-Domain Problems”, “Mesh smoothing for the spectral element method”, “LES of the gas-exchange process inside an internal combustion using a high-order method”), participation in the International Council for Industrial and Applied Mathematics (ICIAM19), the AMD GPU Tutorial and Hackathon meeting in Austin, the ”Summit on Summit/Sierra-III” meeting at NVIDIA, the SciDAC PI meeting, and the NCAR Multicore-9 workshop.

An article highlighting work on the project, “CEED’s Impact on Exascale Computing Project Efforts is Wide-ranging“, was published on the ECP website.

8. CONCLUSION

In this milestone, we developed architecture optimizations and tuning of performant discretization libraries and standards for finite element operators targeting heterogeneous systems. The focus was on delivering optimal data locality and motion, enhanced scalability and parallelism, derived through a number of CEED

backends and software packages. We also delivered performance tuned CEED first and second-wave ECP applications.

The artifacts delivered include performance improvements in CEED's 1st and 2nd wave of applications, and tuned CEED software for various architectures through a number of backends, freely available in the CEED's repository on GitHub. Developed and released were libCEED v0.5, improved GPU support in MFEM, and NekRS using libParanumal. See the CEED website, `http://ceed.exascaleproject.org` and the CEED GitHub organization, `http://github.com/ceed` for more details.

In addition to details and results from the above R&D efforts, in this document we are also reporting on other project-wide activities performed in Q4 of FY19 including: CEED's third annual meeting, initial NeKRS release, collaboration with NVIDIA on Laghos, an overview paper covering latest results on a set of bake-off high-order finite element problems (BPs), and other outreach efforts.

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-758990.

REFERENCES

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

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

[3] Robert W. Anderson, Veselin A. Dobrev, Tzanio V. Kolev, Robert N. Rieben, and Vladimir Z. Tomov. High-order multi-material ALE hydrodynamics. *SIAM J. Sci. Comp.*, 40(1):B32–B58, 2018.

[4] Ellen M Arruda and Mary C Boyce. A three-dimensional constitutive model for the large stretch behavior of rubber elastic materials. *Journal of the Mechanics and Physics of Solids*, 41(2):389–412, 1993.

[5] J. Brown. Efficient Nonlinear Solvers for Nodal High-Order Finite Elements in 3D. *Journal of Scientific Computing*, 45, October 2010.

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

[7] P. Fischer. Analysis and application of a parallel spectral element method for the solution of the Navier-Stokes equations. In C. Canuto and A. Quarteroni, editors, *Spectral and High-Order Methods for Partial Differential Equations, Proc. of the ICOSAHOM 89 Conf., Como, Italy.*, pages 483–491. North-Holland, 1989.

[8] P. Fischer, J. Lottes, and S. Kerkemeier. Nek5000: Open source spectral element CFD solver. http://nek5000.mcs.anl.gov and https://github.com/nek5000/nek5000. 2008.

[9] P Fischer, M Min, T Rathnayake, S Dutta, T Kolev, V Dobrev, J.S. Camier, M Kronbichler, T Warburton, K Swirydowicz, and J Brown. Scalability of high-performance pde solvers. *International Journal of High Performance Computing Applications*, page in print, 2020.

[10] P.F. Fischer. An overlapping Schwarz method for spectral element solution of the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 133:84–101, 1997.

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

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

[13] Gerhard Holzapfel. *Nonlinear solid mechanics: a continuum approach for engineering*. Wiley, Chichester New York, 2000.

[14] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[15] M. Sprague, M. Churchfield, A. Purkayastha, P. Moriarty, and S. Lee. A comparison of Nek5000 and OpenFOAM for DNS of turbulent channel flow. In *Nek5000 Users Meeting*, Argonne National Laboratory, 2010.

[16] Kasia Świrydowicz, Noel Chalmers, Ali Karakus, and Timothy Warburton. Acceleration of tensor-product operations for high-order finite element methods. *arXiv preprint arXiv:1711.00903*, 2017.

[17] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

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