EXASCALE COMPUTING PROJECT

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 31, 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 31, 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 31, 2020	Original	

EXECUTIVE SUMMARY

The main goal of this milestone was to help CEED-enabled ECP applications, including ExaSMR, MARBL, ExaWind and ExaAM, to improve their performance and capabilities on GPU systems like Summit and Lassen/Sierra. In addition, the CEED team also worked to: add and 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. These additional tasks contributed directly to the main CEED-MS34 goal and will also play an important role in CEED's future milestones.

The specific tasks addressed in this milestone were:

- 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, including new releases of many CEED software components;
- Work with CEED applications to improve their GPU performance and capabilities.

All new developments were released under a CEED-3.0 release, and integrated with applications to improve their GPU performance and capabilities.

The artifacts delivered include a number of software releases: CEED-3.0, libCEED-0.6, MFEM-4.1, NekRS-20.0, hipMAGMA-1.0, Laghos-3.0, Remhos-1.0 and GSLIB-1.0.6; 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.

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. 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), ExaConstit – a miniapp for the ExaAM project, as well as ExaWind and E3SM.

The CEED team developed optimization techniques and tuned for performance the CEED software to accelerate the 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 0.6 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, MARBL, E3SM, and ExaAM.

In addition, the CEED team also worked to: add and 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. These additional tasks are documented in the separate sections in this report. They contributed directly to the main CEED-MS34 goal and will also play an important role in CEED's future milestones.

The artifacts delivered include a number of software releases: CEED-3.0, libCEED-0.6, MFEM-4.1, NekRS-20.0, hipMAGMA-1.0, Laghos-3.0, Remhos-1.0 and GSLIB-1.0.6; 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.

2. 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. In this section we describe the efforts 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`)

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. Our first goal was 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 miniapps include high-order finite element solvers for elliptic problems, compressible Navier-Stokes, incompressible Navier-Stokes, and Boltzmannian gas dynamics. Our second goal was 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. Finally we developed a process an improved process to incorporate ongoing and future developments from benchParanumal and libParanumal into the CEED Nek5000, MFEM, and libCEED packages.

2.1 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 2.2.

The pathway to libParanumal kernels integration is based either on the OCCA versions of the benchParanumal 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.

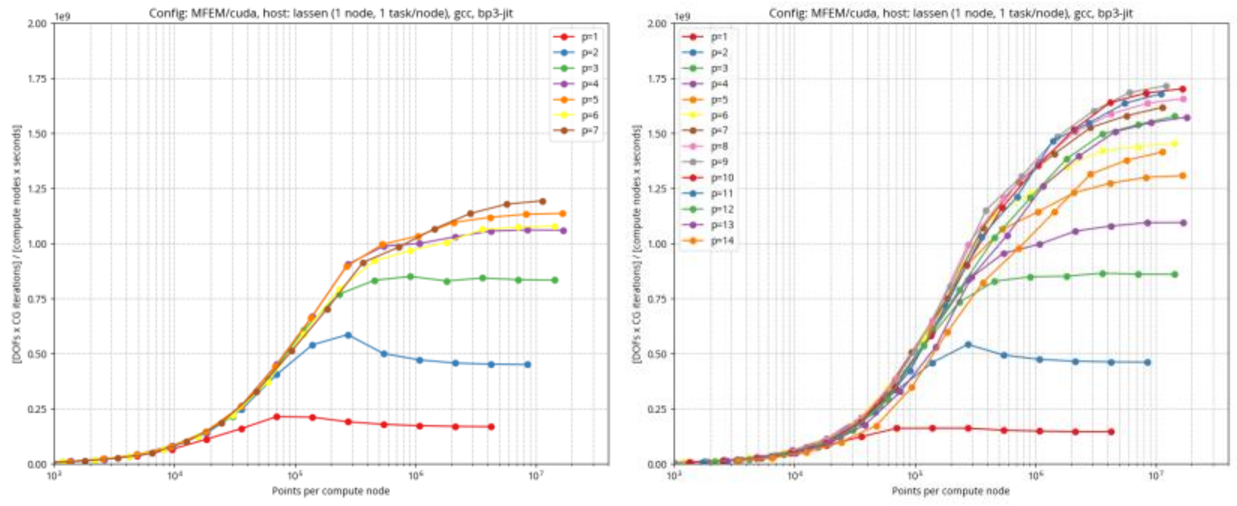


Figure 1: Left: original MFEM diffusion kernel. Right: libParanumal style kernels in MFEM.

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

The MFEM speed up results after the diffusion kernel was transformed in this way are shown in Figure 1. 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 [23]. 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.

2.2 Integrating libParanumal kernels in libCEED

Unlike 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 3 for a comparison before/after on BP3 using libCEED through MFEM.

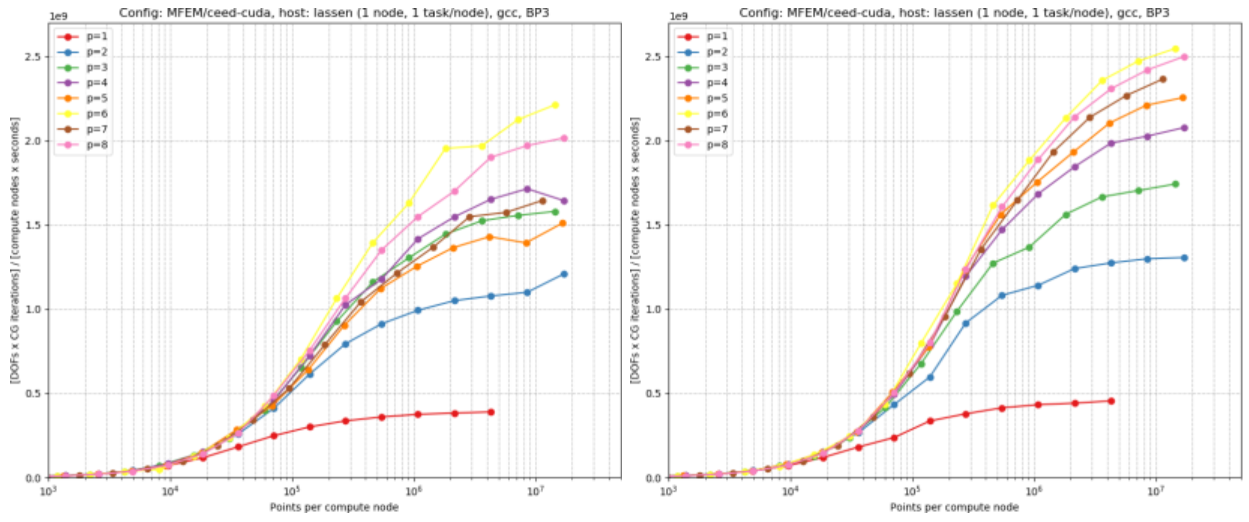


Figure 3: Left: original libCEED diffusion kernel. Right: libCEED diffusion kernel after using libParanumal optimizations.

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.

2.3 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.

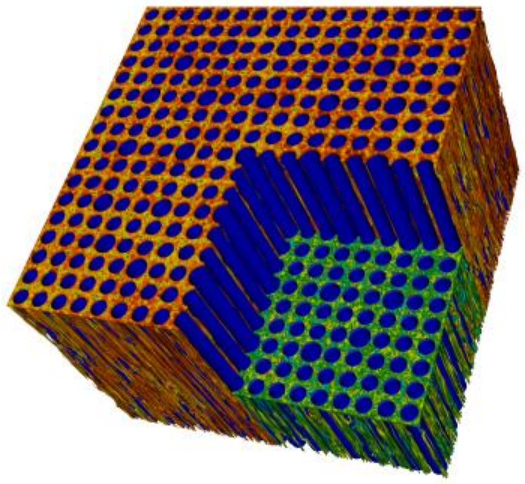


Figure 4: NekRS 17x17 rod-bundle turbulent flow simulation performed on the OLCF Summit system.

Figure 4 demonstrates turbulent flow in 17x17 rod-bundle computed with NekRS on Summit. Figures 5 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 4. 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 5, 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 research group.

Figure 5, 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. See Figure 5, right.

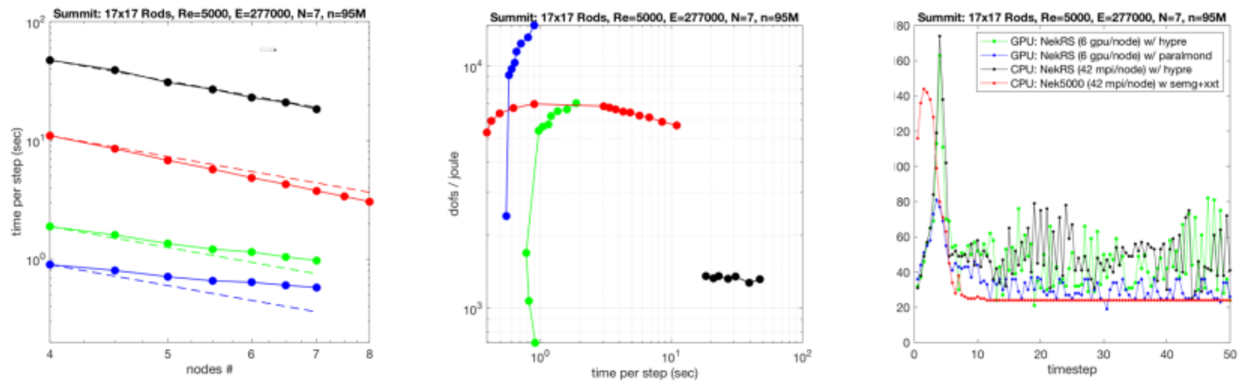


Figure 5: NekRS and Nek5000 performance of GPUs vs. CPUs on Summit for turbulent flow simulations with Re=5000 for a 17x17 rod-bundle geometry using total number of grid points n=95,011,000. Based on timings from Step 11 to 60, time-per-step with ideal scalings shown as dashed lines (left), pressure iterations per step (center), and dofs-per-joule with respect to time-per-step (right) are shown.

2.4 Ongoing integration of benchParanumal and libParanumal kernels into CEED packages

Ongoing and future developments from benchParanumal and libParanumal will continued to be integrated into the CEED Nek5000, MFEM, and libCEED packages as follows:

- 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.

3. CEED-3.0 RELEASE

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

Version CEED-3.0 released on March 31st, 2020 contains 12 integrated packages, ranging from low-level modular libraries to applications, plus the CEED meta-package. We list these packages below, listing some highlights for each of them since the last release.

GSLIB-1.0.6 ↦ Improved testing and portability; enhanced support for “sessions”.

Laghos-3.0 ↦ Unified device support through MFEM-4.1, including CUDA, RAJA, OCCA, HIP, OpenMP, and more. See Section 5.2.2 for more information about this Laghos release.

libCEED-0.6 ↦ New documentation website, numerous new features, backend performance improvements, and examples/miniapps. This release is discussed in detail in Section 4.

MAGMA-2.5.3 ↦ hipMAGMA for AMD GPUs, new symmetric interfaces, and half-precision support. See Section 6.1.3 for more information on the hipMAGMA-1.0 release.

MFEM-4.1 ↦ Improved GPU support, libCEED support, extensions to partial assembly techniques, new solvers, new discretization capabilities, improved test coverage, and new examples/miniapps. Some discussion of backend improvements in MFEM-4.1 can be found in Section 6.1.1.

Nek5000-19.0 ↦ Experimental RANS models, new partitioners, mesh quality smoothing, algebraic multigrid improvements, and improved support for Nek-Nek simulations.

Nekbone-17.0 ↦ Stable release.

NekCEM-c8db04b ↦ Portability and ongoing maintenance.

OCCA-1.0.9 ↦ HIP backend, portability and inlining improvements.

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

PUMI-2.2.2 ↦ Improved testing, examples, and documentation; portability and mesh format completeness.

Remhos-1.0 (REMap High-Order Solver) ↦ New miniapp that solves the pure advection equations used to perform monotonic and conservative discontinuous field interpolation (remap) as part of the Eulerian phase in Arbitrary Lagrangian Eulerian (ALE) simulations. See Section 5.2.1 for more information.

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.

In developing these packages and preparing this integrated release, we strive to follow best practices in open source development (especially as curated by the xSDK project). This involves community aspects like prompt response to issues and review of pull request, advising of new contributors, and clear, up-to-date documentation, as well as technical aspects like portable builds, complete and precise test suites, continuous integration for correctness and code quality metrics, interfaces with a clearly defined software lifecycle, and modular design with opportunities for new contributors.

4. LIBCEED-0.6 RELEASE

libCEED is CEED's low-level API library that provides portable and performant evaluation of high-order operators. Version 0.6 of libCEED was released as part of the the CEED-3.0 release and the CEED-MS34 milestone with a number of new features and performance improvements.

Some of the new features that make it easier to take advantage of the highly optimized libCEED kernels in a wide variety of applications are:

- New expanded documentation at libceed.readthedocs.io.
- 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.
- 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.
- User-provided 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 new libCEED-0.6 release includes a number of improvements in its GPU and CPU backends. These are discussed in detail in Section 2.2 and Section 6.1.3, here is a short summary:

- MAGMA backend was significantly optimized for non-tensor bases.
- User-provided QFunctions using variable-length array pointer constructs can now be used with CUDA backends.
- No-copy optimization in `CeedOperatorApply`.
- Some missing edge cases in the CUDA backend were fixed.
- New backend: `/cpu/self/memcheck/serial`.

An important component of libCEED-0.6 are the greatly expanded set of examples with detailed documentation that can be easily used as a starting point by new users. Some of the notable new examples are discussed briefly in the rest of this section.

4.1 PETSc BPs on the cubed-sphere

This example solves a Laplace problem on the sphere (with a "cubed" mesh) that is relevant to atmospheric modelling and E3SM. The example uses the following coordinate transformations for the computation of the geometric factors: from the physical coordinates on the sphere, denoted by $\overset{\circ}{\mathbf{x}} = (\overset{\circ}{x}, \overset{\circ}{y}, \overset{\circ}{z})$, 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

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

with Jacobian determinant given by

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

We note that in equation (1), the right-most Jacobian matrix $\partial \mathbf{x}/\partial \mathbf{X}_{(3\times 2)}$ is provided by the library, while $\partial \overset{\circ}{\mathbf{x}}/\partial \mathbf{x}_{(3\times 3)}$ is provided by the user with analytical derivatives. In particular, for a sphere of radius 1, we have

$$\overset{\circ}{\mathbf{x}}(\mathbf{x}) = \frac{1}{\|\mathbf{x}\|} \mathbf{x}_{(3\times 1)}$$

and thus

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

For the L^2 projection problems, BP1-BP2, that use the mass operator, the coordinate transformations and the corresponding Jacobian determinant, equation are given by (1) and (2). For the Poisson's problem, BP3-BP6, on the cubed-sphere, in addition to equation (2), the pseudo-inverse of $\partial \overset{\circ}{\mathbf{x}}/\partial \mathbf{X}$ is used to derive the contravariant metric tensor. We begin by expressing the Moore-Penrose (left) pseudo-inverse:

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

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

$$\frac{\partial u}{\partial \overset{\circ}{\mathbf{x}}}_{(1\times 3)} = \frac{\partial u}{\partial \mathbf{X}}_{(1\times 2)} \frac{\partial \mathbf{X}}{\partial \overset{\circ}{\mathbf{x}}}_{(2\times 3)}$$

and thus the weak Laplacian may be expressed as

$$\int_\Omega \frac{\partial v}{\partial \overset{\circ}{\mathbf{x}}} \left(\frac{\partial u}{\partial \overset{\circ}{\mathbf{x}}} \right)^T = \int_\Omega \frac{\partial v}{\partial \mathbf{X}} \underbrace{\frac{\partial \mathbf{X}}{\partial \overset{\circ}{\mathbf{x}}} \left(\frac{\partial \mathbf{X}}{\partial \overset{\circ}{\mathbf{x}}} \right)^T}_{\mathbf{g}_{(2\times 2)}} \left(\frac{\partial u}{\partial \mathbf{X}} \right)^T \tag{4}$$

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,

$$\mathbf{g} = \left(\frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}} \right)^{-1}_{(2\times 2)} \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T_{(2\times 3)} \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}_{(3\times 2)} \left(\frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}} \right)^{-T}_{(2\times 2)} = \left(\frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}^T \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}} \right)^{-1}_{(2\times 2)} \tag{5}$$

where we have dropped the transpose due to symmetry. This allows us to simplify (4) 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`.

4.2 Solid mechanics elasticity miniapp

In this miniapp, 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.

$$\begin{tikzcd}
\text{Finite Strain Hyperelastic} \arrow[r, "\text{constitutive}", "\boldsymbol{S}=\mathsf{C}\boldsymbol{E}"'] \arrow[d, "\text{geometric}"', "\boldsymbol{E}\to\boldsymbol{\epsilon}"] & \text{St. Venant-Kirchoff} \arrow[d, "\text{geometric}"', "\boldsymbol{E}\to\boldsymbol{\epsilon}"] \\
\text{Small Strain Hyperelastic} \arrow[r, "\text{constitutive}", "\boldsymbol{\sigma}=\mathsf{C}\boldsymbol{\epsilon}"'] & \text{Linear Elastic}
\end{tikzcd} \tag{6}$$

The elasticity min-app is controlled via command-line options, which allow the user to use variety of meshes and boundary conditions, see Figure 6. The miniapp 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.

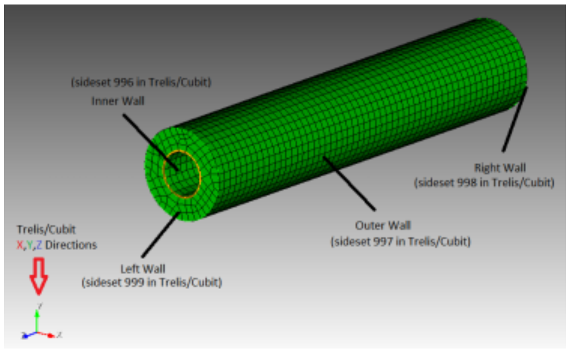

(sideset 996 in Trelis/Cubit)
Inner Wall
Right Wall
(sideset 998 in Trelis/Cubit)
Trelis/Cubit
X,Y,Z Directions
Outer Wall
(sideset 997 in Trelis/Cubit)
Left Wall
(sideset 999 in Trelis/Cubit)
Y
Z

Figure 6: Example of mesh for sample run of libCEED's new elasticity miniapp.

- 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.

Many related solvers can be implemented by composing PETSc command-line options. For additional details, including full documentation of the command line options, example runs and the detailed derivation of the weak formulation, see the online documentation at libceed.readthedocs.io/en/latest/examples/solids.

5. APPLICATIONS GPU 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 in this milestone was 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 effort, we are also reaching out to additional applications in the ECP and also in non-ECP projects.

5.1 ExaSMR

The ExaSMR-oriented effort has focused on improved meshing and extreme scalability for reactor thermal hydraulics problems with Nek5000/NekRS.

The CEED/NEK 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 of interest to the ExaSMR application. The utility generates all-hex meshes by starting with Voronoi cells surrounding each pebble, optimizing these and tessellating each cell facet into quadrilaterals that are projected onto the pebble surfaces. The new approach yields high quality all-hex meshes with element counts that are a factor of six lower than previous approaches based on tet-to-hex conversion, allows the use of more accurate, higher-order, elements for the same resolution (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. Development work was started on 146 pebbles and recently extended to a 1568-pebble simulation with 500,000 elements.

We also have pushed the overall work flow for Nek5000/NekRS to support over 60 million 7th-order elements, *the largest number of spectral elements in production runs to date.* We present strong- and weak-scaling studies for 17×17 rod-bundle simulations. Improved performance on GPUs and CPUs using the latest repo versions of NekRS and Nek5000 during are also demonstrated in this section.

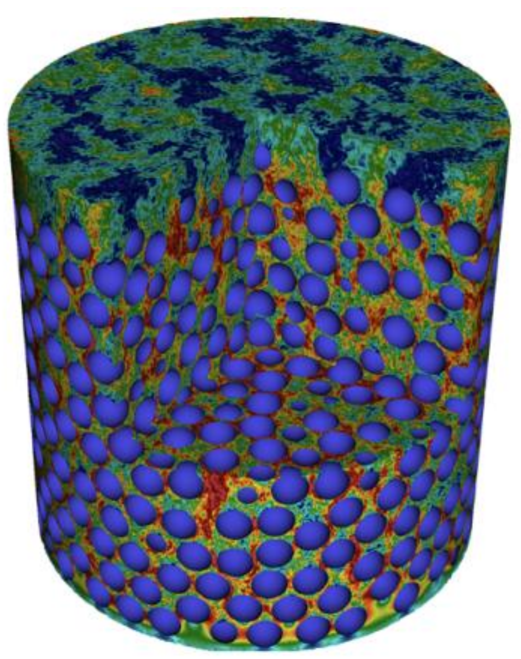


Figure 7: Velocity distribution for flow past 1568 pebbles at $Re = 10,000$ simulated with NekRS using 66 V100 GPUs on Summit. The all-hex mesh comprises $E = 524,386$ elements of order $N = 7$ ($n = 179,864,398$).

5.1.1 GPU Strong-Scaling Performance on Pebble Beds Reactor Simulations

Figure 7 demonstrates the NekRS capability to simulate complex flows. Shown is the velocity distribution for flow at Reynolds number $Re = 10,000$ in a 1568-pebble bed that has been run on 66 GPUs on Summit. The total number of gridpoints is $n = 179,864,398$ with $E = 524,386$ elements of order $N = 7$. Figure 8, left, shows the corresponding number of GMRES iterations per timestep in pressure solve for different node counts, while Figure 8, right shows the accumulated wall-clock time. 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) with corresponding number of grid points per GPU of (n/gpu) $2.7M$, $1.3M$, and $680K$. Table 1 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 shown in Section 2.3. We achieve 75% parallel efficiency using 1.3M points per GPU and 52% using $680K$ points per GPU.

Figure 9 demonstrates a performance comparison between the tet-to-hex and new all-hex mesh for the 146-pebble case at approximately the same resolution. The tet-to-hex mesh has $n = 23M$ ($E = 365844$, $N = 4$) while the new all-hex mesh uses $n = 21M$ with ($E = 63132$, $N = 7$). Figure 9(left) shows the number of (Nek5000) GMRES iterations in pressure solve for varying timestep sizes for each of the two meshes. The tet-to-hex mesh requires ≈50 iterations per timestep with $\Delta t = .001$ (convective time units), whereas the all-hex mesh requires ≈10–20 iterations while using a three-times larger timestep. The benefits of the improved mesh are borne out in the long-time integration results of Figure 9(right). For the same wall-clock time, the new meshing approach realizes roughly an order-of-magnitude increase in simulated time. The next section provides more detail on the new all-hex meshing approach.

5.1.2 Novel All-Hex Meshing Strategies for Dense-Packed Spheres

Flow through beds of randomly-packed spheres is encountered in many science and engineering applications. 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)[8], which uses local tensor-product bases on curvilinear brick

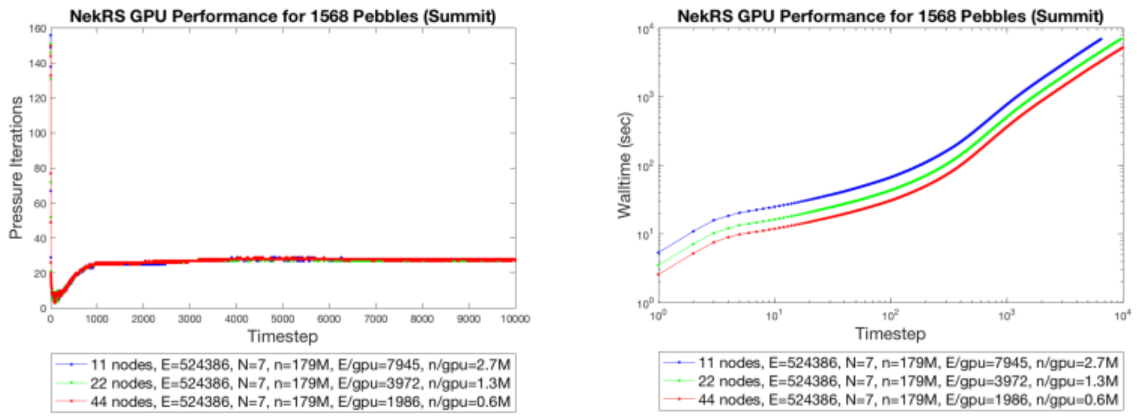


Figure 8: NekRS GPU performance on 11, 22, and 44 nodes using total 66, 132, and 264 GPUs of Summit, respectively, for the 1568-pebbles mesh in Figure 7. GMRES iterations per timestep in pressure solve (left) and accumulated walltime measured per timestep (right) are shown.

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 1: NekRS Strong-scaling performance on Summit GPUs, measured at 1000 and 5000 timesteps for 1568 pebbles in Figure 7 with $E = 365844$.

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. Alternative tet-supporting high-order formulations (Dubrinov) 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., [11]). This strategy has been pursued in a recent article by Yuan *et al.* [27]. 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 [9]. 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.

The CEED Nek team developed novel meshing strategies for generating high-quality hexahedral element mesh 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

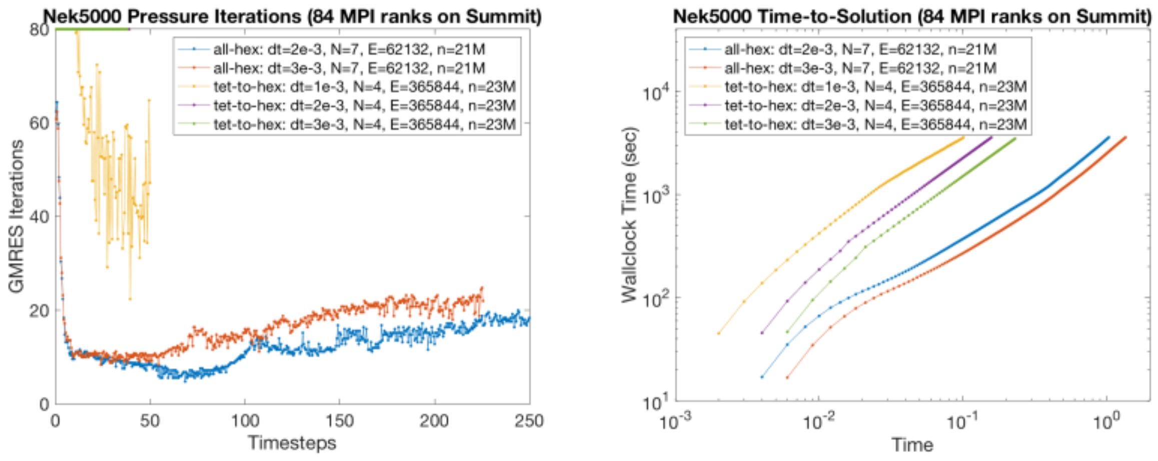


Figure 9: GMRES iterations (left) and time-to-solution (right) comparison between all-hex and tet-to-hex meshes. Meshes represent 146 pebbles for a pebble-beds reactor geometry.

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[8] and simulations are performed using Nek5000 [10]. 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 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 introduction 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.

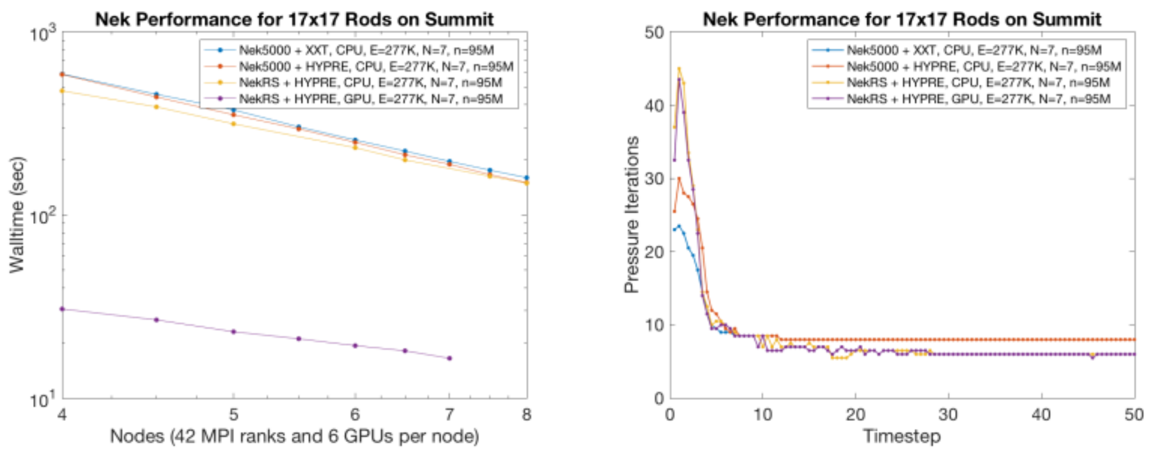


Figure 10: NekRS/Nek5000 CPU/GPU strong-scaling performance on Summit. Accumulated walltime (left) measured using 101–200 timesteps and the GMRES iterations per timestep in pressure solve are shown. 17×17 rod-bundle turbulent flow simulations ($Re = 5000$, $E = 277000$, $N = 7$ (total 95 millions grids).

5.1.3 NekRS GPU/CPU Strong-Scaling Performance on 17×17 *Rod-bundle Simulations*

Figure 10 demonstrates recent strong-scaling performance on Summit's CPUs and GPUs for 17×17 rod-bundle turbulent flow simulations for $Re = 5000$ using a mesh $E = 277000$ and $N = 7$ (total 95 millions grid points). Figure 10, left, shows the walltime measured per timestep using 100–200 timesteps demonstrating improved strong-scale behaviors than previous baseline studies in Figure 4. Timings on NekRS GPU runs show 10× speedup, compared to those on NekRS CPU and Nek5000 CPU runs. Figure 10, right, demonstrates the GMRES iteration behaviors per timestep in pressure solve.

Tables 2–3 demonstrate larger sizes of weak-scaling performance tests. Table 2 describes the configuration of testing set for 17×17 rod-bundle reactor geometry with the number of spectral elements increased by the number of levels in z by increasing the domain size in $[0, z_{\max}]$. We keep the number of element as approximately 12000 per GPU (total ∼4.1M points) and approximately 1715 per CPU (total $n =$∼588K points). Table 3 demonstrates weal-scaling performance on Summit, using 6 GPUs and 42 CPUs per node. Accumulated walltime is measured using 101–200 timesteps for 17×17 rod-bundle turbulent flow simulations for $Re = 5000$ running NekRS on GPUs and CPUs and Nek5000 on CPUs.

Figure 11, left, demonstrates averaged walltime per timestep, using the same data in Table 3. The results show good weak-scaling on NekRS and Nek5000 on CPU performance. NekRS/Nek5000 CPU/GPU weak-scaling performance using 4–846 nodes on Summit, using up to a maximum number of 5076 GPUs and 35532 CPUs. The number of elements varies from $E = 277000$ to $E = 60,940,000$ (largest number of spectral elements in production runs, having total 20 billions of grid points using $N = 7$). NekRS increases timings slightly as the number of nodes increases. The averaged walltime per timestep is 0.3–0.6 seconds for NekRS GPU runs, 5–6 seconds for NekRS CPU runs, and 3–4 seconds for Nek5000 CPU runs. NekRS GPU performance keeps 8–10× speedup from 4 to 846 nodes, compared to NekRS and Nek5000 CPU performances. Figure 11, right, shows the GMRES iterations per timestep in pressure solve for NekRS CPU and GPU runs and Nek5000 CPU runs. This is ongoing efforts while the data for the case of $E = 120M$ is not available yet.

5.2 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.

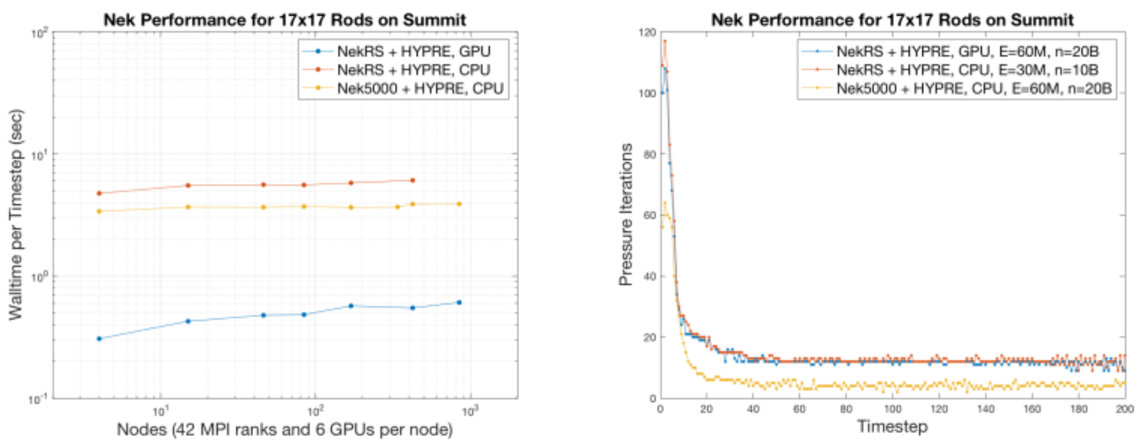


Figure 11: NekRS/Nek5000 CPU/GPU weak-scaling performance using 4–846 nodes on Summit, using up to a maximum number of 5076 GPUs and 35532 CPUs. The number of elements varies from $E = 277000$ to $E = 60,940,000$ (largest number of spectral elements in production runs, having total 20 billions of grid points using $N = 7$). Averaged walltime per timestep (left), measured using 101–200 timesteps, and GMRES iterations per timestep in pressure solve (right) are shown, using the data in Tables 2–3 presenting 17×17 rod-bundle turbulent flow simulations $Re = 5000$.

z-levels	$z_{\max}$	E	nodes	#GPU	E/GPU	#CPU	E/CPU
10	0.455	277000	4	24	11541.67	168	1648.81
40	1.82	1108000	15	90	12311.11	630	1758.73
120	5.46	3324000	46	276	12043.48	1932	1720.50
220	10.01	6094000	84	504	12091.27	3528	1727.32
440	20.02	12188000	169	1014	12019.72	7098	1717.10
880	40.04	24376000	338	2028	12019.72	14196	1717.10
1100	50.05	30470000	423	2538	12005.52	17766	1715.07
2200	100.1	60940000	846	5076	12005.52	35532	1715.07

Table 2: NekRS/Nek5000 CPU/GPU weak-scaling performance configurations for 17×17 rod-bundle with elements of order $N = 7$. The problem size is increased by increasing the number of layers of elements in z with increasing domain size in $[0, z_{\max}]$, keeping the number of elements approximately 12000 per GPU (~4.1M points/GPU) and 1715 elements per CPU (~588K points/CPU). Timing results are shown in Table 3.

5.2.1 *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, namely, the discretization methods described in the articles [6, 5, 7, 13, 14].

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.

z-levels	$z_{\max}$	E	nodes	#GPU	#CPU	NekRS/GPU 100 steps (sec)	NekRS/CPU 100 steps (sec)	Nek5000 CPU 100 steps (sec)
10	0.455	277000	4	24	168	30.68	475.68	338.65
40	1.82	1108000	15	90	630	42.67	551.97	367.97
120	5.46	3324000	46	276	1932	47.74	560.82	366.79
220	10.01	6094000	84	504	3528	48.33	557.73	371.01
440	20.02	12188000	169	1014	7098	56.95	579.35	365.75
880	40.04	24376000	338	2028	14196	–	–	367.59
1100	50.05	30470000	423	2538	17766	54.88	608.69	389.07
2200	100.1	60940000	846	5076	35532	60.81	–	389.78

Table 3: NekRS/Nek5000 CPU/GPU weal-scaling performance on Summit using 6 GPUs and 42 CPUs per node for 7th-order spectral elements. Accumulated walltime (sec) measured per Navier-Stokes timestep using 101–200 timesteps are shown for 17×17 rod-bundle turbulent flow simulations for $Re = 5000$. The configuration of the testing problems are detailed in Table 2.

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 is shown in Figure 12. Remhos-1.0 was released as part of CEED-3.0.

5.2.2 Laghos miniapp improvements

By utilizing the new capabilities of MFEM-4.1, the performance and codebase of Laghos were 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 CEED-MS25) 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.

These improvements are available in the latest release, Laghos-3.0, which was released as part of CEED-3.0.

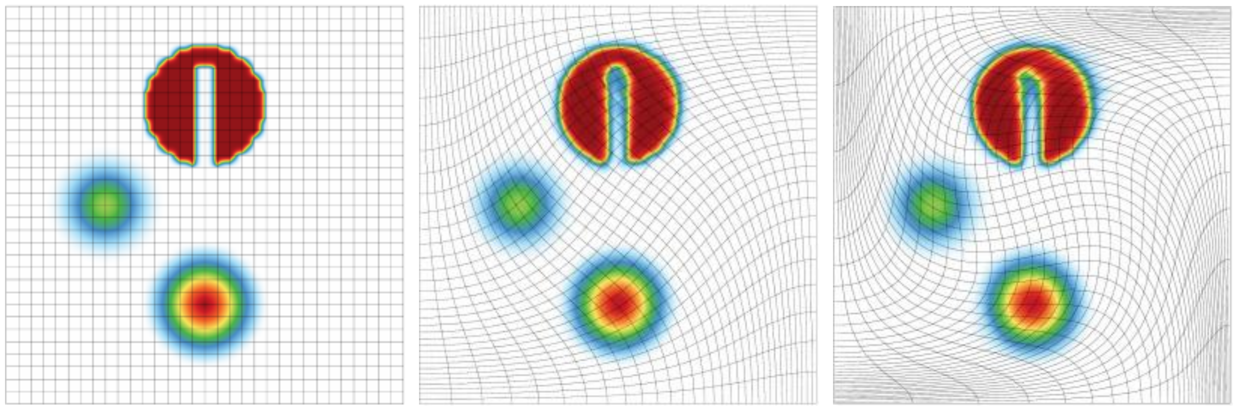


Figure 12: Example from Remhos demonstrating a high-order monotonic and conservative remap calculation with different amounts of mesh distortion.

5.2.3 GPU work in MARBL

The focus of the GPU efforts in MARBL has been on 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 4×. Preliminary results have demonstrated performance improvements between 2× and 10× 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 13. 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.

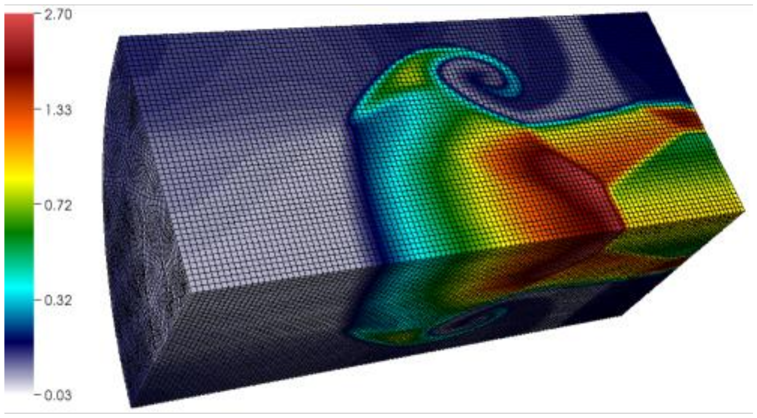
The global CG mass matrix inversion is fully ported on GPUs. The device mass action kernels are very similar to the ones in MFEM, but currently live in the BLAST code, as the previous MFEM implementation did not support non-constant integral coefficients. Since such coefficients has been recently implemented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



32 GPUs + 32 MPI Procs
ALE type: Eulerian, T = 4.98

Figure 13: MARBL result obtained on 32 NVIDIA Tesla V100 GPUs at LLNL.

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.

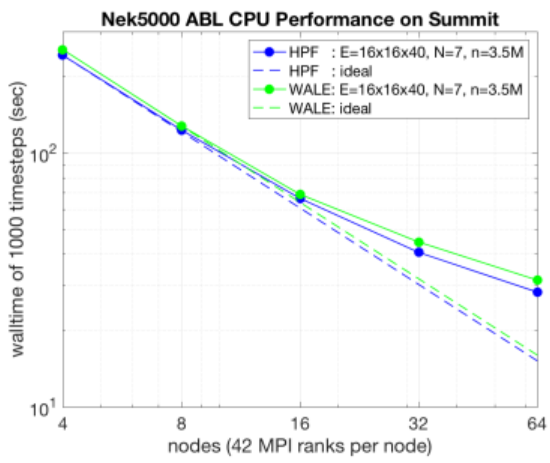
Using the MFEM infrastructure, the MARBL developers have developed custom GPU code to populate the sparsity of the advection sparse matrix, thus achieving LO and HO implementations for the GPU. It is expected that this approach will be improved significantly by the work in Remhos, as it contains matrix-free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

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.

5.3 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 far field 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 grid points than second-order schemes [22].

This work involves efforts by Ananias Tomboulides (AUTH, ANL) from the CEED/Nek team and Matthew Churchfield, Shreyas Ananthan, Michael Sprague from the ExaWind team at NREL. The plan is to develop a bake-off study toward the cross-verification and validation of our LES results and corresponding wall models, 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 result as well as a newly developed uRANS approach, k-τ model.



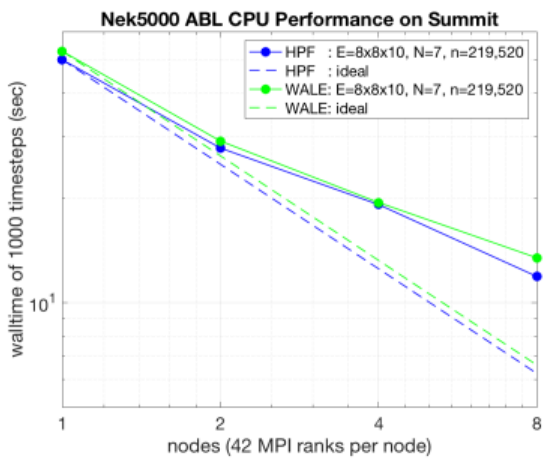


Figure 14: Nek5000 strong-scaling performance for ABL simulations ($Re = 50000$) using $E = 10240$ and $E = 640$ with $N = 7$ on Summit. (Left) Running on 672 cores (16 nodes, 42 cores per node) using 5226 points per core, it reaches strong-scale limit (90% efficiency) with averaged timing of 0.07 seconds 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.

5.3.1 *Atmospheric Boundary Layers for Wind Applications*

Efficient simulation of atmospheric boundary layer flows (ABL) 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 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 [16]. 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. It is important to perform a high-fidelity LES simulation of stable ABL flows as well as development of improved wall models.

5.3.2 *Atmospheric Flows Test Problems and Baseline Performance on Summit*

In collaboration with ExaWind researchers, the CEED Nek team has focused on developing reliable high-fidelity LES-model implementations for stable ABL flows and improved wall models. Our implementation of wall models, for smooth or rough walls, based on log-law behavior is performed by imposing traction boundary conditions specified through boundary integral terms ([12, 17, 21]). This approach differs from the one used in finite volume codes and requires the specification of the traction boundary location in wall units. A well-documented stably stratified atmospheric boundary layer benchmark problem that can be used for these purposes, i.e. for model and code comparison, is the Global Energy and Water Cycle Experiment (GEWEX) Atmospheric Boundary Layer Study (GABLS) ([2, 3, 1]). 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 Nek team was able to develop SEM algorithms and validate the implementation of the ABL model in Nek5000. Figure 14 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 14, 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 14, 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 Nek 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/Nek and Exawind teams have helped to solidify the problem definition and feasible approaches to robust implementations.

5.3.3 *Newly Developed Regularized uRANS Approach,* $k - \tau$ *Model*

We have developed, implemented, and validated a regularized version of the k–ω RANS model (also referred to as $k - \omega'$ that avoids the singularity in ω at wall boundaries through the use of asymptotic expansions [24]. More recently, CEED Nek team has implemented an SEM-based version of the k–τ model [15] that does not suffer from singular behavior at walls. From preliminary verification tests, we found that the $k - \tau$ model gives the same results with the $k - \omega$ model but its main advantage is that it does not rely on the wall-distance function or its derivatives and the terms appearing in the right-hand-side of the model equations are bounded close to walls. We will continue investigating this model more extensively and to use it further for the study of more complex flows.

Currently we have tested preliminary studies on the $k - \omega$ and $k - \tau$ models in a number of benchmark internal flows, such as turbulent channel flow, flow past a backward-facing step, as well as external flows, such as flow past a wind turbine blade and the NACA0012 airfoil at 0° and 10° angle of attack (aoa) [19]. For external flows, we have investigated the performance of several limiters, commonly used in the RANS literature, for the production terms in the $k - \omega$ and the $k - \tau$ equations as well as for the eddy viscosity. Two versions of the model were evaluated that differ in the values of some of the model coefficients [25], [26] (noted as kw98 and kw06, respectively); the 2006 model includes an additional cross-diffusion term. As an example, the performance of the models was tested for the flow past a NACA0012 airfoil for Reynolds number of 6 millions at aoa = 0° and 10°, with varying free-stream conditions; fully converged results for the drag and lift coefficients were obtained even for zero free-stream values of k and results for all cases agree very well with benchmark solutions obtained by Nek5000. (see Figure 15 and Tables 4–5). We are currently extending this effort into `NekRS` to support running on GPUs.

5.4 ExaAM

The ExaAM project couples 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 were chosen by the project as the appropriate

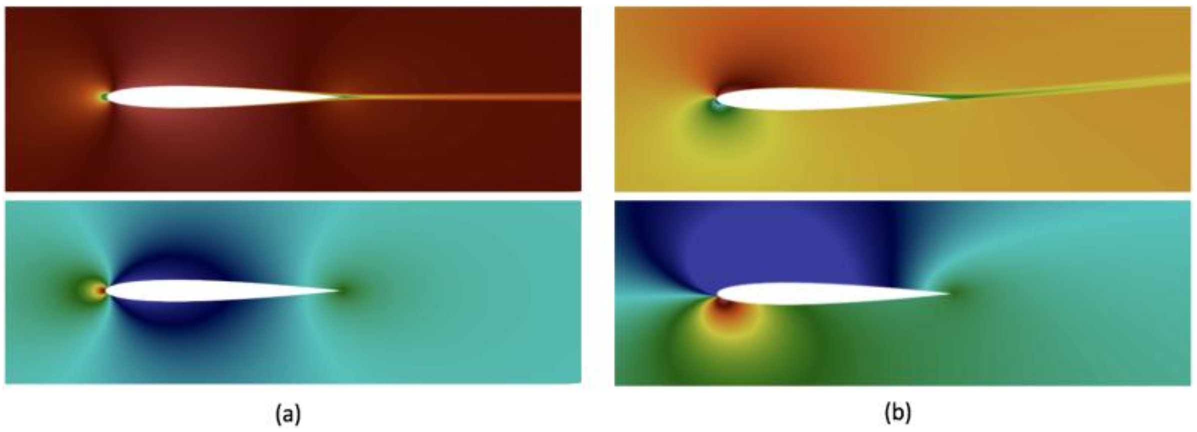


Figure 15: RANS models into Nek5000: k-τ and wall function models, allowing coarser resolutions NACA0012 RANS simulations for airfoil, $Re = 6M$. (a) velocity component v_x (top) and pressure (bottom) profiles for aoa = 0°. (b) velocity component v_x (top) and pressure (bottom) profiles for aoa = 10°.

Table 4: Drag and lift coefficients (aoa=0). Experimental data: NASA-TM-4074 [18].

	Nek5000 results				References		Exper.
model	kw98 (N=7/N=11)	kτ98 (N=7)	kw06 (N=7/N=11)	kτ06 (N=7)	CFL3D	FUN3D	
drag	0.00872 / 0.00843	0.00842	0.00861/0.00833	0.00832	0.00854	0.00837	∼0.0081
lift	±1E-5/±1E-5	1.55E-5	±1E-5/±1E-5	1.21E-5	∼0	∼0	∼-0.01

Table 5: Drag and lift coefficients (aoa=10). Experimental data: NASA-TM-4074 [18].

	Nek5000 results				References		Exper.
model	kw98 (N=7)	kτ98 (N=7)	kw06 (N=11)	kτ06 (N=7/N=9)	CFL3D	FUN3D	
drag	-	0.01507	0.01391	0.01468/0.01432	0.01259	0.01297	∼ 0.012
lift	-	1.0582	1.0639	1.0592/1.0609	1.0958	1.1012	∼ 1.075

computational approach for the microstructure-to-properties aspect of the overall ExaAM problem. Since none of the existing crystal mechanics-based codes were suited to the ExaAM needs, the ExaAM team is creating an application, ExaConstit, specifically for local property analysis. This development is a collaboration between ExaAM and the CEED teams.

ExaConstit's first release was 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 and is freely available on GitHub at `https://github.com/LLNL/ExaConstit`.

5.4.1 The ExaConstit miniapp

The main development work in the FY20 Q2 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. This partial assembly formulation follows the CEED finite element decomposition framework, as exemplified for example by the libCEED user interface. It allows ExaConstit to take advantage of tensor contraction operations in-order to achieve the lowest number of FLOP count and memory storage requirements. Outside of this reformulation, ExaConstit was refactored to take advantage of the various parallelization abstraction layers introduced in MFEM v4.0 to allow various compute and material kernels to run on the GPU.

Some of the development activities between ExaConsit and CEED are summarized below:

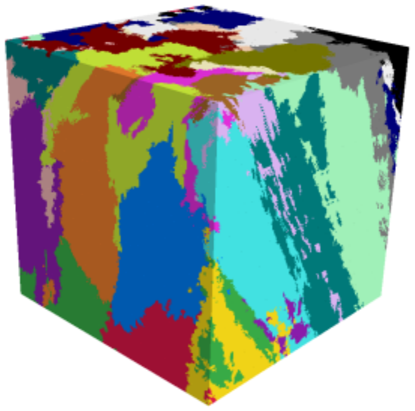


Figure 16: ExaConstit: ExaCA generated grain microstructure that approximates AM processes for a 8e6 element mesh.

- 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

Over the course of the ExaAM activity, the plan is for ExaConsit to make use of the MFEM capabilities for higher-order finite elements with the goal 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.

5.4.2 Results from a demonstration problem

As a part of a recent ExaAM milestone, we consider test problems centered around an ExaCA-generated [ExaCA/FY19Q1] microstructure for 316L [FY18Q3] stainless steel. One model being used is from ExaCMech 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 16. The mesh is $200 \times 200 \times 200 = 8 \times 10^6$ elements and is a discretization of a polycrystal with 380 grains.

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. On 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.

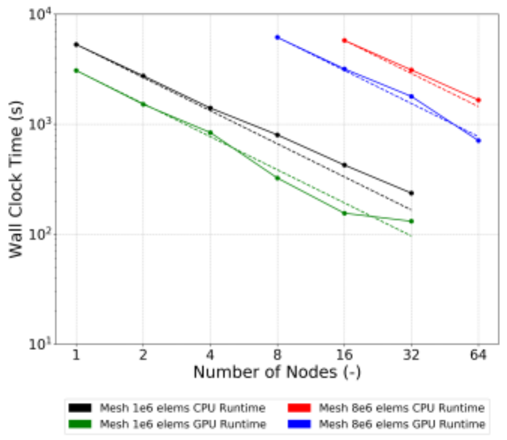


Figure 17: ExaConstit: Strong scale study conducted on Summit ExaCMech model for CPU and GPUs.

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 6: ExaConstit: 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 7: ExaConstit: Timings of strong scale study conducted on Summit for 8e6 element mesh.

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 17 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 18 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.

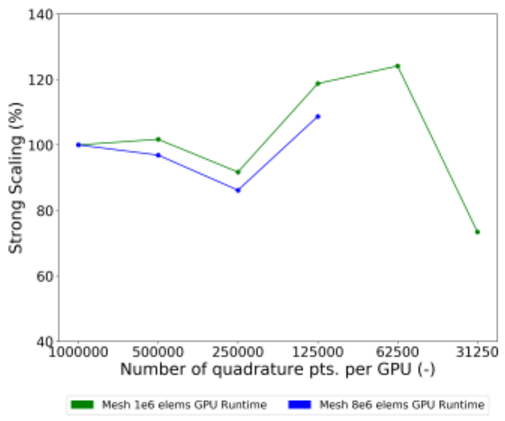


Figure 18: ExaConstit: GPU strong scale study as a function of data residing on each GPU.

5.5 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 Division 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.

- **Other applications.** The CEED team is also participating in ECP-organized outreach activities to industry, international partners, and project from NASA and non-ECP DOE labs.

6. ADDITIONAL HARDWARE AND PROGRAMMING MODELS

This section describes some recent efforts in adding/improving support for additional hardware and programming models in the CEED software components.

6.1 Support for 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-Compute Interface for Portability (AMD HIP).

CEED researchers are 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; 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.

6.1.1 MFEM backends improvements

MFEM version 4.0 introduced new GPU capabilities and support for hardware accelerators such as CUDA, OCCA, libCEED, RAJA and OpenMP. The newly releases 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.

6.1.2 Porting libParanumal

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 initial 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 hierarchy. For NVIDIA GPUs much of this information has been released by NVIDIA or reverse engineered by the micro-benchmarking 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 publicly 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 significantly 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 hierarchy 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, see the CEED-MS32 report.

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 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 truly performance portable despite different characteristics of the GPUs from the different vendors and their respective tools ecosystems.

In Figure 19 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

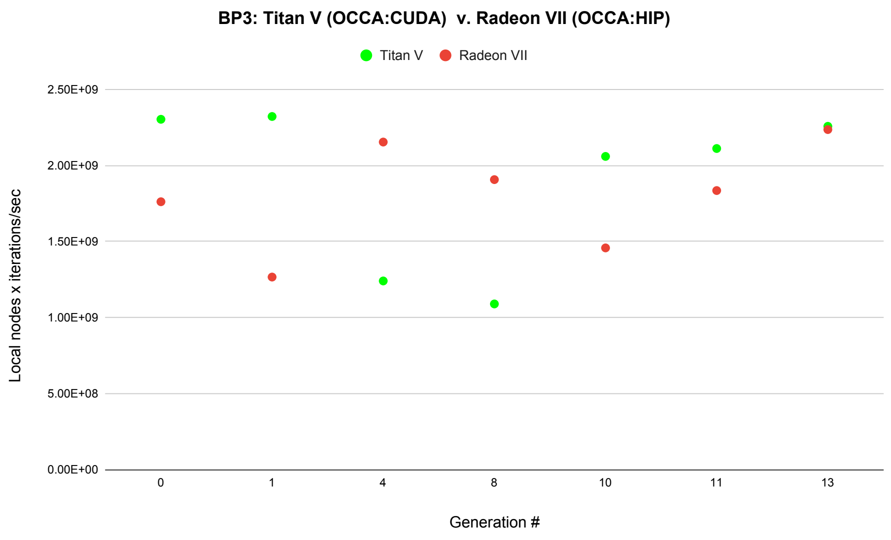


Figure 19: Generations of BP3 implementations tuned to attain high throughput for both the AMD Radeon VII and NVIDIA Titan V GPUs.

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.

6.1.3 hipMAGMA

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.

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, see Section 3.

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 20 illustrates the performance

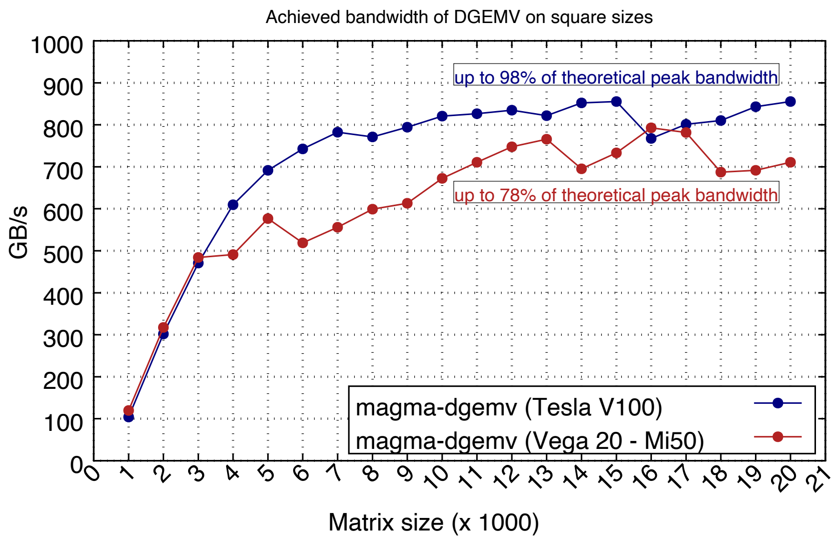


Figure 20: The matrix-vector multiply benchmark in double precision (DGEMV). Results are shown for the Tesla V100 GPU (CUDA 9.1) and the MI50 GPU (HIP 3.0).

of a 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 (today) 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 21 shows a compute-bound benchmark; the matrix-matrix multiplication kernel (DGEMM). Similar to the DGEMV benchmark, the achievable (today) 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 (assembly GEMMs) outperform the (CUDA/HIP GEMM) kernels available in MAGMA [20]. 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 next. Figure 22 shows the performance of the batched DGEMM [4] 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

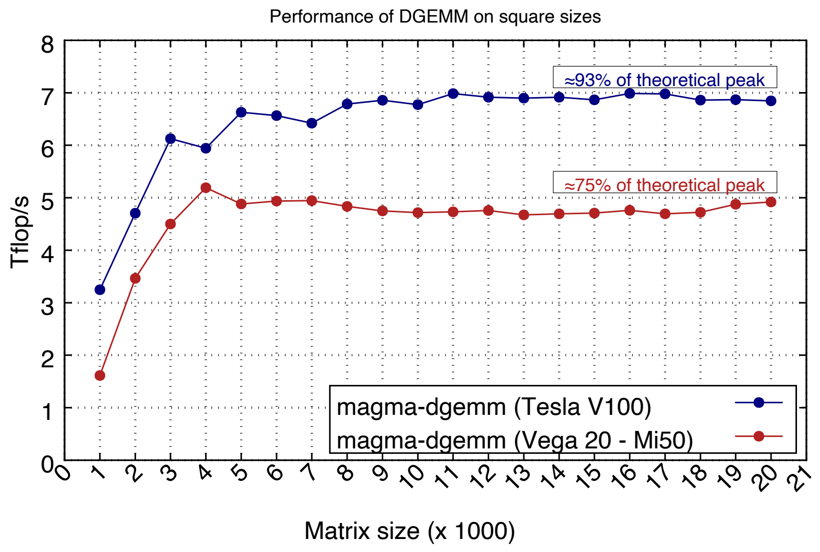


Figure 21: The matrix-matrix multiply benchmark in double precision (DGEMM). Results are shown for the Tesla V100 GPU (CUDA 9.1) and the MI50 GPU (HIP 3.0).

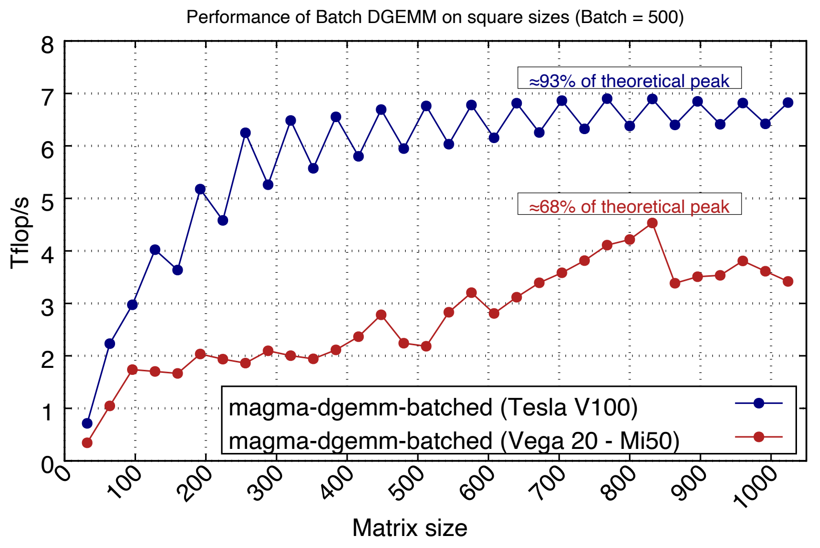


Figure 22: The batched matrix-matrix multiply benchmark in double precision (DGEMM). Results are shown for the Tesla V100 GPU (CUDA 9.1) and the MI50 GPU (HIP 3.0).

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.

BLAS Performance for Non-tensor Basis Action in libCEED. The non-tensor basis action in libCEED can be performed using the GEMM kernel. 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 23. Ideally, a single DGEMM operation would be enough to operate at the GPU peak performance. However, the relatively large 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.

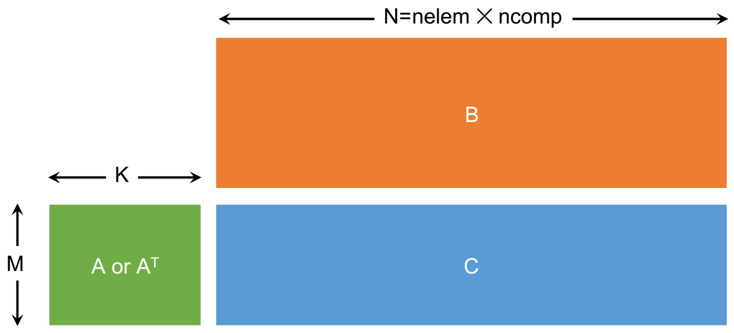


Figure 23: Shape of the DGEMM operation for the non-tensor basis action in libCEED.

Figure 24 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.

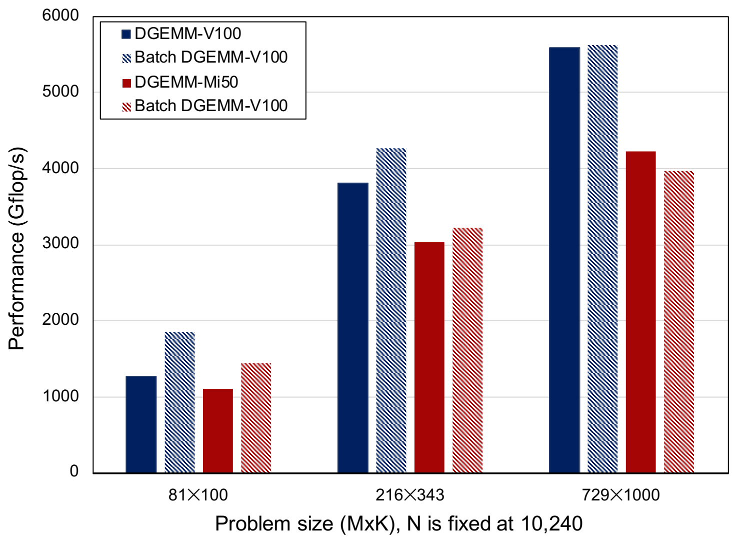


Figure 24: Performance for the non-tensor basis action in libCEED for typical problem sizes using DGEMM and Batch DGEMM.

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 development. 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`.

6.2 Support for Intel GPUs

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.

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 8: 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

Additionally, the CEED team member Thilina Rathnayake (UIUC Ph.D. student) is currently an intern at Intel working on ports of Nekbone (the CEED BP5 bake-off problem) to Intel iGPU Gen9, which is a proxy for Aurora development work.

We recall that BP5 is Jacobi-preconditioned conjugate gradient iteration applied to a spectral-element-(SE-) based Poisson problem on a tensor-product array of hexahedral elements. Full geometric factors are included so that the problem mimics a complex domain with deformed elements. For BP5, the quadrature points are the same as the Gauss-Lobatto-Legendre (GLL) nodal points, so no interpolation is required to evaluate the SE-based matrix-vector product, $A\underline{u}$. Neglecting communication for the assembly, the leading-order flop count for a pth-order expansion in 3D is $12(p+1)^4 + 18(p+1)^3$ operations per element and the number of memory references is $7(p+1)^3$ per element.

Thilina has ported BP5 to OpenMP and has run this on Nvidia V100s and Intel iGPUs. He has also developed an OpenCL variant that has been tested on Intel iGPU Gen9. Current plans are to optimize on the Gen9 and then on the Aurora simulator. Performance tuning has addressed both BK5 (just the local matrix-vector product with no communication for assembly) and *gslib*, which is the portable gather-scatter library native to the Nek code suite. Current plans are to develop an OCCA port once the OpenCL port is complete.

7. OTHER PROJECT ACTIVITIES

7.1 Advanced Simulation Workshop at ANL

Members of CEED and ECP attended the Advanced Simulation Workshop 2020 `http://www.asw2020.org`, a workshop organized by Misun Min at Argonne to bring together researchers from a variety of application areas to review and discuss current trends in scientific simulation. Topics included HPC, biological fluid mechanics, geodynamos, oil spill modeling in the ocean, combustion, nuclear reactor modeling, trends in GPU architectures, fundamentals of turbulence, and reduced order modeling. CEED/ECP attendees included Tzanio Kolev, Tim Warburton, Ananias Tomboulides, Paul Fischer, Misun Min, Elia Merzari, Aleks Obabko and Andrew Siegel. Posters were presented by CEED-supported Ph.D. students Pedro Bello-Maldonado, Malachi Phillips, and Nicholas Christensen, all from UIUC.

7.2 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 released on GitHub since August, 2019, and a very first official release, `NekRS` v20.0, is announced on March 31, 2020. NekRS has been used for large scale simulations. A version of 2019 was ported to Aurora development system to get baseline performance on Intel Gen9, compared to Nvidia V100 – these results as well as an initial baseline performance on Summit using 17×17 rod geometries are demonstrated in Section 2.3. Recently NekRS has been focusing on simulations on ExaSMR problems with pebble beds and 17×17 rod geometries as demonstrated in Section 5.1 (using a repo version in March 2020).

7.3 `gslib` v1.0.6 Release

A new version of `gslib` `https://github.com/Nek5000/gslib` was released as v1.0.6. Major features and improvements in this new release include addition of multisession findpts, backwards-compatible changes, and some bug fixes.

7.4 Outreach

CEED researchers were involved in a number of outreach activities, including a successful breakout session on high-order methods and application at the ECP second annual meeting in Houston, organizing a minisymposium on numerical PDEs at the AMS JMM20 meeting, participation in the JOWOG34 meeting, the El Capitan COE kickoff at LLNL, the ECP industry deep dive workshop and a ECP-UK telecon, as well as 6 papers, 2 presentations and 1 poster. The ECP podcast interview about CEED titled "Helping Applications Leverage Future Computing Architectures with First-Rate Discretization Libraries" was posted 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.

In addition, the CEED team also worked to: add and 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.

The artifacts delivered include a number of software releases: CEED-3.0, libCEED-0.6, MFEM-4.1, NekRS-20.0, hipMAGMA-1.0, Laghos-3.0, Remhos-1.0 and GSLIB-1.0.6; 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.

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-808458.

REFERENCES

[1] A large eddy simulation study of a quasi-steady, stably stratified atmospheric boundary layer. *Journal of the Atmospheric Sciences*, (57):1052, 2000.

[2] An intercomparison of large-eddy simulations of the stable boundary layer. *Boundary-Layer Meteorology*, (118):247–272, 2006.

[3] Adding complex terrain and stable atmospheric condition capability to the openfoam-based flow solver of the simulator for on/offshore wind farm applications (sowfa). Germany, March 20–21, 2013.

[4] Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. Performance, design, and autotuning of batched gemm for gpus. In *The International Supercomputing Conference (ISC High Performance 2016)*, Frankfurt, Germany, 2016-06 2016.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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

[12] H. Grotjans and F. Menter. Wall functions for general application cfd codes. In *Proc. the 4th Computational Fluid Dynamics Conference, ECCOMAS 98*, pages 1112–1117, New York, 1998. J. Wiley & Sons.

[13] 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.

[14] 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.

[15] G Kalitzin, A.R.B. Gould, and J.J Benton. Application of two-equation turbulence models in aircraft design. *AAIA*, 96:0327, 1996.

[16] N.D. Kelley. Turbulence-turbine interaction: The basis for the development of the turbsim stochastic simulator. Technical Report TP-5000-52353, National Renewable Energy Laboratory, 2011.

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

[18] Charles L. Ladson. Effects of independent variation of mach and reynolds numbers on the low-speed aerodynamic characteristics of the naca 0012 airfoil section. Technical Report NASA-TM-4074, L-16472, NAS 1.15:4074, NASA Langley Research Center, Hampton, VA, United States, 1988.

[19] Misun Min, Ananias Tomboulides, Paul Fisher, Elia Merzari, Dillon Shaver, Javier Martinez, Haomin Yuan, and YuHsiang Lan. Nek5000 enhancements for faster running analysis. Technical Report ANL/MCS-TM-384, Argonne National Laboratory, Lemont, IL, United States, 2019.

[20] Rajib Nath, Stanimire Tomov, and Jack Dongarra. An improved magma gemm for fermi graphics processing units. *Int. J. High Perform. Comput. Appl.*, 24(4):511–515, November 2010.

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

[22] 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.

[23] 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.

[24] A. Tomboulides, M. Aithal, P. Fischer, E. Merzari, A. Obabko, and D. Shaver. A novel numerical treatment of the near-wall regions in the k-ω class of the rans models. *International Journal of Heat and Fluid Flow*, 72:186–199, 2018.

[25] D.C. Wilcox. *Turbulence Modeling for CFD.* DCW Industries, La Canada, CA, 1998.

[26] D.C. Wilcox. Formulation of the k-ω turbulence model revisited. *AAIA Journal*, 46(11):2823–2838, 2008.

[27] 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.