ECP Milestone Report

Performance tuning of CEED software and 1st and 2nd wave apps

WBS 2.2.6.06, Milestone CEED-MS32

Stanimire Tomov
Ahmad Abdelfattah
Valeria Barra
Natalie Beams
Jed Brown
Jean-Sylvain Camier
Veselin Dobrev
Jack Dongarra
Yohann Dudouit
Paul Fischer
Ali Karakus
Stefan Kerkemier
Tzanio Kolev
YuHsiang Lan
Elia Merzari
Misun Min
Aleks Obabko
Scott Parker
Thilina Ratnayaka
Jeremy Thompson
Ananias Tomboulides
Vladimir Tomov
Tim Warburton

October 2, 2019

ECP Milestone Report
Performance tuning of CEED software and 1st and 2nd wave apps
WBS 2.2.6.06, Milestone CEED-MS32

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

October 2, 2019

ECP Milestone Report
Performance tuning of CEED software and 1st and 2nd wave apps
WBS 2.2.6.06, Milestone CEED-MS32

Approvals

Submitted by:

Tzanio Kolev, LLNL CEED PI	Date

Approval:

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

Revision Log

Version	Creation Date	Description	Approval Date
1.0	October 2, 2019	Original	

EXECUTIVE SUMMARY

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

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

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

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

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

In addition to details and results from the above R&D efforts, in this document we are also reporting on other project-wide activities performed in Q4 of FY19 including: CEED's third annual meeting, MFEM GPU kernels scaling improvements, libCEED 0.5 and NekRS releases, an overview paper covering latest results on a set of bake-off high-order finite element problems (BPs), and other outreach efforts.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

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

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

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

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

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

2. LIBCEED-0.5 RELEASE

libCEED version 0.5 was released in September 2019. Notable features of this release are improved GPU backends, single-source Q-functions for CPU and GPU implementations, some improvement in vectorized instructions for Q-function code compiled for CPU (by using `CeedPragmaSIMD` instead of `CeedPragmaOMP`), implementation of a Q-function gallery and identity Q-functions, and the expansion of the PETSc benchmark problems to include unstructured meshes handling. Two of these advances, single-source Q-functions and improved GPU backend performance, are described below.

In addition to the 0.5 release, the libCEED team collaborated with Oana Marin, from the PETSc team at ANL on expanding the PETSc examples to support unstructured meshes. This new capability will allow the implementation of examples on complex geometries. Ongoing collaborations also include joint effort by the libCEED team and Ken Jensen from CU Boulder so that some of the PHASTA application features can be ported to libCEED, including SUPG stabilization techniques.

2.1 Single-source Q-functions

Single-source Q-functions were developed as a result of collaborative effort between the libCEED team at CU Boulder and the MFEM team at LLNL. Users can define Q-functions in a single source code independently of the targeted backend with the aid of a new macro `CEED_QFUNCTION` to support JIT (Just-In-Time) and CPU compilation of the user provided Q-function code. To allow a unified declaration, the Q-function API has undergone a slight change: the `QFunctionField` parameter `ncomp` has been changed to `size`. This change requires setting the previous value of `ncomp` to `ncomp*dim` when adding a QFunctionField with eval mode `CEED_EVAL_GRAD`.

The single-source Q-functions allowed us to develop a Q-function gallery. In the initial Q-function gallery, mass and Poisson Q-functions are available for 1D, 2D, and 3D. These Q-functions come in pairs for partial assembly, with separate Q-functions to setup the geometric factors and apply the action of the PDE. The initial Q-function gallery also includes an identity Q-function, to aid in construction of more general operators, such as restriction and prolongation operators for multigrid.

2.2 State-of-the-art GPU Performance

In this section we describe a sequence of libCEED GPU backends of increasing complexity and performance that were developed by the CEED team over the last year. In order to achieve the highest performance, GPU codes require almost all problem-dependent constants, such as dimensions, array sizes, loops bounds, to be known at *compilation* time. Due to its modular nature, libCEED provides most of these dimensions at *runtime*, e.g. order of the method, number of quadrature points, number of fields. Thus, a natural way to develop performant high-order kernels is to take advantage of JIT compilation which fixes the problem-dependent constants (at runtime) that the compiler sees. We use JIT via NVIDIA's NVRTC in all of the backends described below.

Due to the relatively low arithmetic intensity, the performance of our algorithms is governed largely by saturating the memory bandwidth via a number of techniques such as: using registers to minimize the information exchange between threads (*CUDA-reg* backend), minimizing the use and synchronization of shared memory (*CUDA-shared* backend), and kernel fusion to minimize device memory access (*CUDA-gen* backend).

2.2.1 Reference Backend (CUDA-ref)

Our first backend, *CUDA-ref*, naturally followed the logical decomposition of a finite element operator exposed by the libCEED interface, i.e. element restriction/prolongation, basis interpolation and its transpose, application of the Q-function at quadrature points. Each of these operations corresponds to a GPU kernel, see Fig. 1 to see the profiling trace for the kernels of a 2D diffusion operator. This reference backend was focusing on providing a feature-complete backend for GPU devices that is easy to read. It constitutes the foundation of all the other backends and is the basis for our GPU optimization efforts.

The consistency of the memory between host and device is one of the main features of the reference backend and is reused by all other backends. Data is copied from host to device and from device to host only when necessary, the backend keeps tracks of the location of the data and moves the data only if required, completely transparently to the user. The reference backend uses JIT compilation to inline problem-dependent constants, allowing loop unrolling and mapping arrays on registers. JIT is also used to generate the user-provided Q-function by inlining the user code inside a kernel generated at runtime.

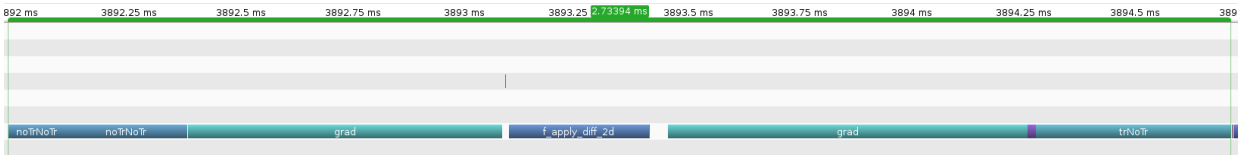

Figure 1: Profiling trace of a 2D diffusion operator for the *CUDA-ref* backend on a NVIDIA V100 GPU.

2.2.2 Register Backend (CUDA-reg)

The second backend, *CUDA-reg*, focused on optimizing the most expensive kernels: basis interpolation and its transpose. The parallelization strategy in this backend was to use one GPU thread per element. This allows each thread to be completely independent and so we avoid data transfer between threads. However, due to the high register usage resulting from this approach the performance at high orders is limited. Comparing Fig. 2 to the reference backend Fig. 1 shows a significant improvement of the basis interpolation/transpose kernels.

However, the irregularity of the memory access, required by the parallelization strategy, has a negative impact on the prolongation and restriction kernels.

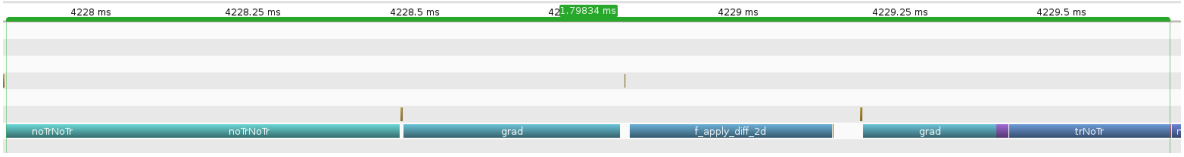

Figure 2: Profiling trace of a 2D diffusion operator for the *CUDA-reg* backend on a NVIDIA V100 GPU.

2.2.3 Shared Backend (CUDA-shared)

Our third backend, *CUDA-shared*, builds on the lessons learned from the *CUDA-reg* backend. In order to exploit efficiently the GPU resources, the parallelization strategy here is to use a block of threads per element. This means that the *CUDA-shared* backend has to use shared memory to exchange data between threads. While we generally observe better performance than *CUDA-reg* in 3D (when the latter spills out of registers), the use of shared memory introduces a potential new bottleneck since its latency is much higher than register latency. This can result in lower device memory transactions being issued per cycle, resulting in lower device memory throughput, and ultimately impacting performance. The *CUDA-shared* backend is designed to avoid this problem by carefully balancing the information stored in registers and shared memory. Comparing Fig. 3 to the *CUDA-reg* backend Fig. 2 shows an improvment of the basis interpolation and transpose kernels, but more importantly this performance improvment is not limited to low orders.

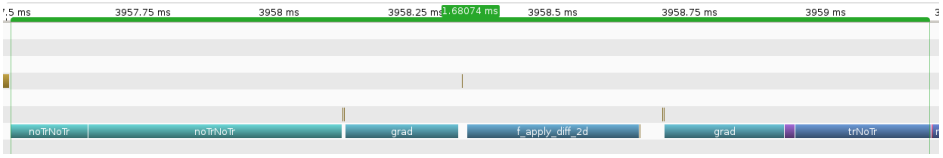

Figure 3: Profiling trace of a 2D diffusion operator for the *CUDA-shared* backend on a NVIDIA V100 GPU.

2.2.4 Fused Backend (CUDA-gen)

The fourth backend, *CUDA-gen*, fuses all the different kernels used by the previous backends into one kernel. This is our best performing backend, which achieves more than 2.5 GDOF/s performance on a V100 GPU, see Fig. 5. This represents about 5× speedup compared to the *CUDA-ref* backend. The best performance is achieved for higher orders (e.g. $p = 6$) and large problem sizes (about 10M dofs per GPU). We note also that high-order methods offer about 5× speedup compared to first order with the same number of degrees of freedom.

Since data movement is the critical aspect to optimize, a natural strategy is to call only one kernel to keep data in registers, instead of having a sequence of kernels reading from and writing to device memory. The challenge is that some of the inner-most parts of the operator (e.g. the Q-function) is not known in advance, meaning that the *CUDA-gen* backend has to generate code on the fly according to the user specification and use JIT compilation to generate a single kernel. The strategy in this backend is to reuse the optimized kernels from the *CUDA-shared* backend but to use them as device functions, generating at runtime a code using these functions to reflect the user operator, and finally generate with JIT compilation a single kernel that applies the operator. By eliminating all the intermediary device memory accesses, this backend provides an important performance improvement (compare Fig. 4 to Fig. 3).

A close collaboration between the libParanumal and MFEM teams led to additional improvements in the *CUDA-gen* backend that minimized register use for gradient evaluation. The optimization experience

gathered by the libParanumal team for specific kernels was successfully ported to libCEED and provided a significant performance improvement. The code automatically generated by libCEED is achieving almost identical performance as the highly specialized libParanumal kernels. The *CUDA-gen* backend achieves close to roofline performance on benchmarked problems (BP1 and BP3) up to order 11, and excellent performance for higher orders (see Fig. 5).

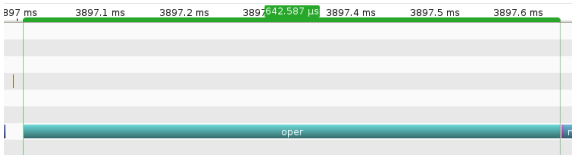

Figure 4: Profiling trace of a 2D diffusion operator for the *CUDA-gen* backend on a NVIDIA V100 GPU.

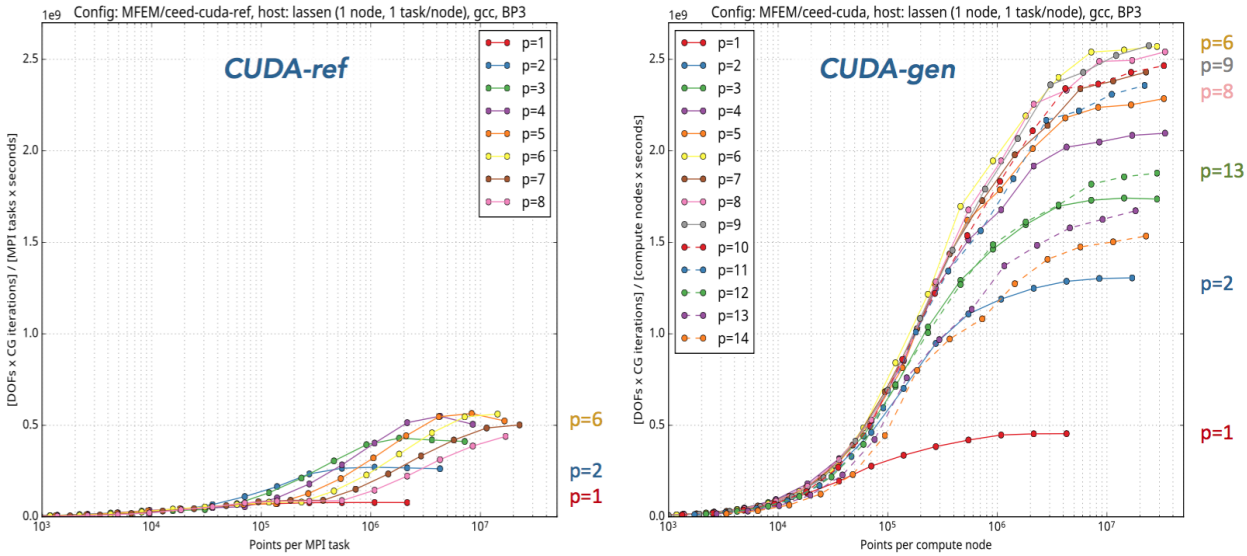

Figure 5: *CUDA-ref* and *CUDA-gen* backends performance for 3D BP3 on a NVIDIA V100 GPU.

2.3 MAGMA backend

For almost a decade, the MAGMA library has been providing highly-optimized and tuned linear algebra kernels that run efficiently on GPUs. In particular, MAGMA focuses on implementing standard BLAS and LAPACK operations that can take advantage of the compute power of GPUs. The MAGMA backend of the CEED library relies on performing efficient BLAS operations on batches of small matrices (i.e., Batched BLAS). In addition, the MAGMA backend also provides a number of customized GPU kernels that fulfill certain requirements of libCEED. Such custom kernels are required either due to: (1) a core functionality cannot be expressed in terms of a standard BLAS operation, or (2) the use of standard Batched BLAS kernels is not expected to deliver a good performance, especially when a sequence of BLAS calls is being performed on a small input data set (e.g., finite elements).

2.3.1 Batched BLAS

Most of the computational workload in a libCEED backend can be represented as performing a sequence of tensor contractions over the finite elements. Each contraction can be expressed in terms of matrix

multiplications. In most applications, the sizes of these multiplications are quite small, meaning that a single contraction does not provide enough computational workload for massively parallel architectures like GPUs. However, the relatively large number of finite elements in an application leads to a large number of independent contractions, hence a huge number of independent multiplications. In the context of dense linear algebra, such workloads are called "batch workloads". It has been proven that batched linear algebra requires parallelization across problems to be implemented on the kernel level (e.g., using independent thread-blocks in the same kernel), not on the runtime level (e.g., using concurrent CUDA streams to launch independent kernels). The former produces significant performance gains against the latter. As an example, Figure 6 shows the performance gains achieved by using a batched matrix multiply kernel against parallelized calls to a regular (non-batched) kernel. The performance advantage of the batch kernel is very significant for small-to-medium sizes. It is obvious that, at some relatively large size, there is gain in using a batch kernel. This is a point where a single operation is large enough to saturate the underlying hardware. However, such sizes are too large to appear in FEM codes.

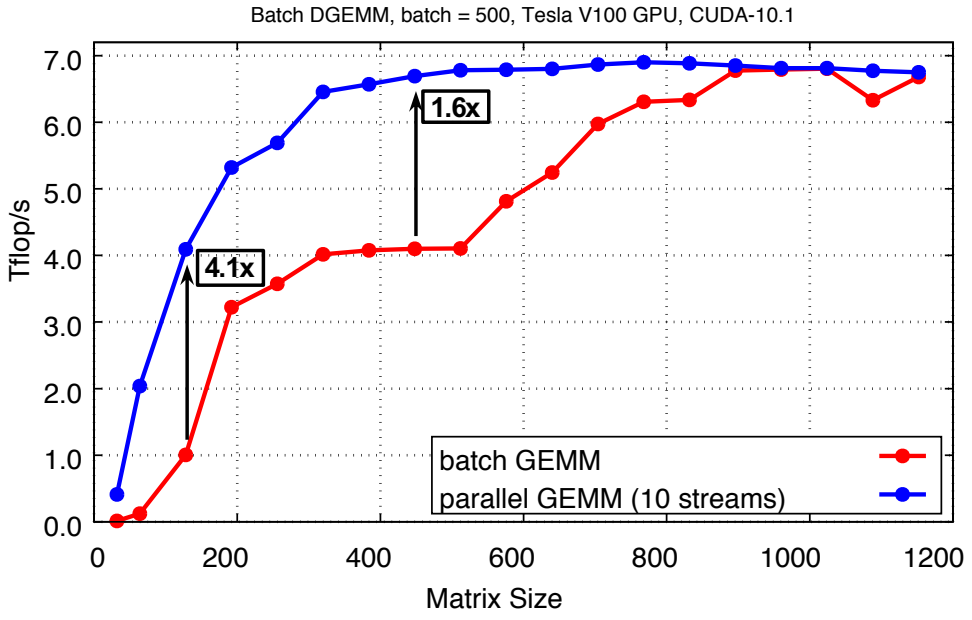

Figure 6: Performance comparison between a batch GEMM kernel, and a parallelized regular GEMM kernel

For this milestone we continued our interactions with vendor and the community in order to design an extension to the BLAS standard (the Batched BLAS), enabling users (and in particular CEED software and users) to perform thousands of small BLAS operations in parallel applications whilst making efficient use of their hardware [1].

2.3.2 Tensor contractions using Batched BLAS

As mentioned before, a single tensor contraction in libCEED can be expressed in terms of a batched GEMM. For example, the reference tensor contraction code in libCEED looks like the following:

```
int
CeedTensorContractApply_Ref(
        CeedTensorContract contract,
        CeedInt A, CeedInt B, CeedInt C, CeedInt J,
        const CeedScalar *restrict t, CeedTransposeMode tmode, const CeedInt Add,
        const CeedScalar *restrict u, CeedScalar *restrict v)
{
  CeedInt tstride0 = B, tstride1 = 1;
  if (tmode == CEED_TRANSPOSE) {
    tstride0 = 1; tstride1 = J;
```

```
  }

  if (!Add)
    for (CeedInt q=0; q<A*J*C; q++)
      v[q] = (CeedScalar) 0.0;

  for (CeedInt a=0; a<A; a++)
    for (CeedInt b=0; b<B; b++)
      for (CeedInt j=0; j<J; j++) {
        CeedScalar tq = t[j*tstride0 + b*tstride1];
        for (CeedInt c=0; c<C; c++)
          v[(a*J+j)*C+c] += tq * u[(a*B+b)*C+c];
      }
  return 0;
}
```

Recalling that GEMM is expressed as $C = \alpha A \times B + \beta C$, this code can be expressed as a batched GEMM routine. There are various ways to do this in the CEED tensor contractions and we implemented parametrized versions for different choices, and in preparation of the final code for autotuning. One specific batching choice related to the above example is when, e.g., the batch size is taken to be the `A` loop. In this case, the input vector u can be viewed as a batch of `C`×`B` matrices, while the output vector v is a batch of `C`×`J`. The input t is viewed as a single matrix of size `B`×`J` if `tmode` is equal to `CEED_NOTRANSPOSE` and `J`×`B` otherwise. The scalar α is always set to one. The conditional accumulation to v can be simply performed by setting β to one if `Add` is `true`, or zero otherwise. Each matrix in u will be multiplied by the same matrix (t) to produce the corresponding matrix in v. The equivalent call in MAGMA becomes:

```
magma_dgemm_batched( transU, transT, C, J, B,
                     alpha, dU_array, ldu,
                            dT_array, ldt,
                     beta,  dV_array, ldv, A, queue);
```

The arguments in this call are set as follows:

- `transU` is always set to `MagmaNoTrans`;
- `transT` is `MagmaNoTrans` or `MagmaTrans`, based on `tmode`;
- `ldu` = `ldv` = `C`, and `ldt` = `B` (no-trans) or `J` (trans);
- `dU_array[i]` = `u + i×C×B`, `i = 0, 1, ···, A-1`;
- `dV_array[i]` = `v + i×C×J`, `i = 0, 1, ···, A-1`;
- `dT_array[i]` = `t`, regardless of `i`.

The advantage of using batched GEMM is to ensure performance portability, e.g., as the batched BLAS API is becoming a standard and the kernels are always subject to continuous improvement, optimization, and tuning across architectures by vendors and other open-source efforts. As part of the MAGMA backend in CEED we developed, tuned, and released through MAGMA a highly optimized batched GEMM kernel that specifically target small sizes. For example, on a V100 GPU for square matrices of size 32, we achieve an execution rate of about 1,600 gigaFLOP/s in double-precision arithmetic, which is 95% of the theoretically derived peak for this computation on a V100 GPU.

However, most of the basis actions in libCEED require applying a sequence of tensor contractions, not just one. While the use of a standard kernel like GEMM achieves performance portability, a sequence of GEMM calls leads to redundant memory traffic, by writing the result of a GEMM operation to the GPU main memory, and reading it back for the subsequent GEMM operation, and so on. Considering that the GEMM sizes are very small, such a redundant memory traffic causes performance drops, since the multiplication remains bounded by the memory bandwidth, and not by the peak FLOP rate of the hardware.

2.3.3 Tensor contractions using Customized Batched BLAS

In order to overcome the limitations of using the standard batched GEMM kernels, we completed the development of the MAGMA backend to offer all customized GPU kernels that are needed in order to implement the basis actions in the latest libCEED release. The following properties are common among such kernels:

1. The most inner building block is a matrix multiplication code that has been demoted from the kernel level to the device routine level;
2. The input and output vectors for a single element are cached in the shared memory of the GPU;
3. A tensor contraction operation is performed by calling the GEMM device routine as many times as required;
4. A certain basis action, therefore, can be performed entirely in shared memory, leading to an optimal data locality and motion;
5. All kernels use a tunable two-dimensional thread configuration;
6. One thread block handles one basis action for a single element.

The biggest gain in performance comes from the savings in memory bandwidth. Thanks to kernel fusion, the new basis action kernels have more arithmetic intensity than a standard GEMM kernel, which leads to a better performance on the GPU. An an example, Figure 7 shows the performance results for a sequence of GEMM calls that is typical in the MFEM code. The horizontal axis shows the size of the element as well as the number of elements (i.e., the size of the batch). Although the standard MAGMA GEMM kernel delivers a better performance than cuBLAS, it is the fused MAGMA kernel that delivers a significantly improved performance, running up to 100× faster than the cuBLAS-based contraction.

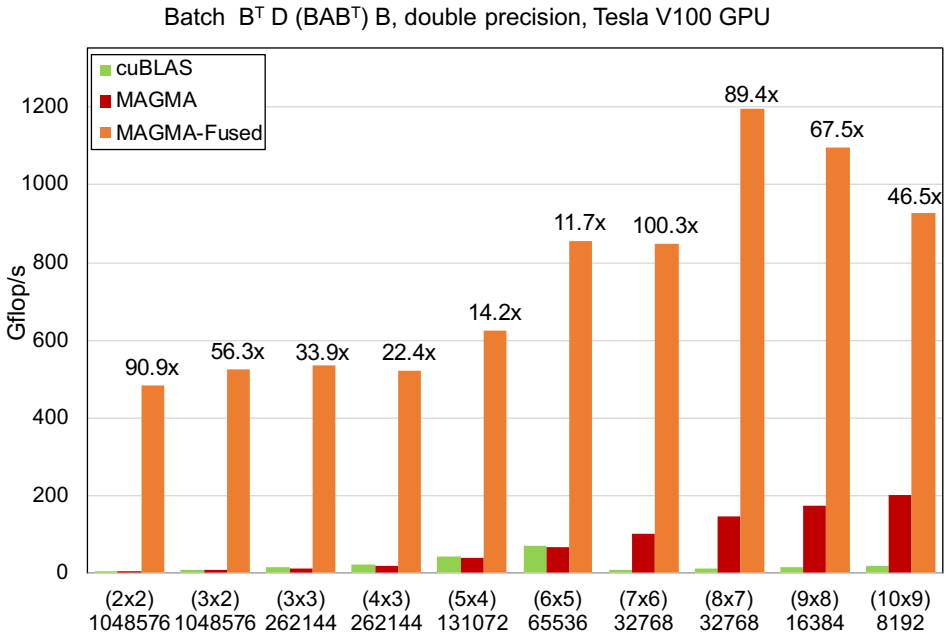

Figure 7: Performance of the typical operational sequence in MFEM.

The effect of fusing kernels in the MAGMA backend using Batched BLAS is similar to the fusing effect in the other CEED backends, e.g., illustrated on Figure 4 vs. Figure 3.

2.3.4 Optimization and Tuning

From the experience gained through developing the MAGMA library, we usually observe increased performance sensitivity to tuning parameters when operating on small sizes. This is why the customized kernels have several tuning parameters that control different aspects of the kernel. We recognize two different categories of parameters:

1. **Compile-time parameters**. Such parameters must be known at compile-time to enable a number of optimizations, such as register blocking and loop unrolling. The customized basis action kernels are written using CUDA C++ templates for this purpose. The most important compile-time parameters are the number of nodes in one dimension (`P1d`), and the number of quadrature points in one dimension (`Q1d`). These two parameters appear in the inner-most loops, and are used to specify the size of some register arrays.

2. **Run-time parameters**. These parameters still affect the performance of the kernel, but they are not necessarily needed at compile-time. When running tuning-sweeps over this type of parameters, re-compilation is not required, which reduces the time required to finish the tuning sweep. An example of such parameter is the thread configuration of our basis action kernels. This parameter affects the occupancy of the developed kernels (e.g. more threads means less throughput on the multiprocessor level). However, it does not affect the unrolled loops or register variables, so it can be defined at run-time. Depending the on the element sizes, as well as the number of elements, this parameter should be set to ensure the best occupancy, which leads to faster throughput per multiprocessor.

For this milestone, we tuned the MAGMA backend kernels based on the order P of the finite elements used, as well as a number of other tuning parameters. In contrast to the CUDA backends, the MAGMA backend does not use runtime compilation (NVRTC). Instead, optimized codes for particular P values (and other constants stemming from the choice) get pre-compiled.

3. APPLICATIONS PERFORMANCE IMPROVEMENTS

3.1 ExaSMR

The strong collaboration between CEED and ExaSMR teams has focused on solving strategies for large, complex pebble geometries ($148 \sim 6000$ pebbles) and 17×17 rod bundles that go up to a mesh size of 120 millions of spectral elements. We address the current status of our simulation capabilities and remaining challenges.

3.1.1 Premilinary JFNK Results for Pebble Beds ($E = 5M$)

In previous milestone, we studied heat transfer for rod bundles [13] by solving the steady linear advection-diffusion equation using GMRES with Schwarz-preconditioned fast diagonalization method. In this milestone, JFNK method involving pseudo-timestepping is considered as relatively more stable approach and is applied to solve steady thermal problems for very complex geometries. In this framework, the equation to be solved by the JFNK steady-state solver is

$$\frac{\partial \overline{T}}{\partial t} + \overline{u}_j \frac{\partial \overline{T}}{\partial x_j} = \frac{\partial}{\partial x_j}\left((\alpha + \alpha_t)\frac{\partial \overline{T}}{\partial x_j}\right), \tag{1}$$

where $\overline{u}_j$, α and α_t are given.

Our JFNK method is introduced in [15] for solving drift-diffusion systems, coupled with Poisson equation describing electric potential. This approach has been directly extended to Navier-Stokes and RANS models. In [15], we provide detailed studies on the choices of parameters that can assure accuracy and efficiency. Our spatial discretization is based on the spectral-element method using curve-fitted hexahedral elements. We apply a backward difference formula (BDF) to the semidiscrete form $\frac{d\underline{u}}{dt} = \underline{f}(\underline{u})$ and build a system of nonlinear equations $\underline{g}(\underline{u})$ expressed with the time step size $\Delta\tau$ and $\underline{f}(\underline{u})$ during the pseudo time-transient process. Then we solve the resulting nonlinear system $\underline{g}(\underline{u}) = 0$ by the Newton method until the solution

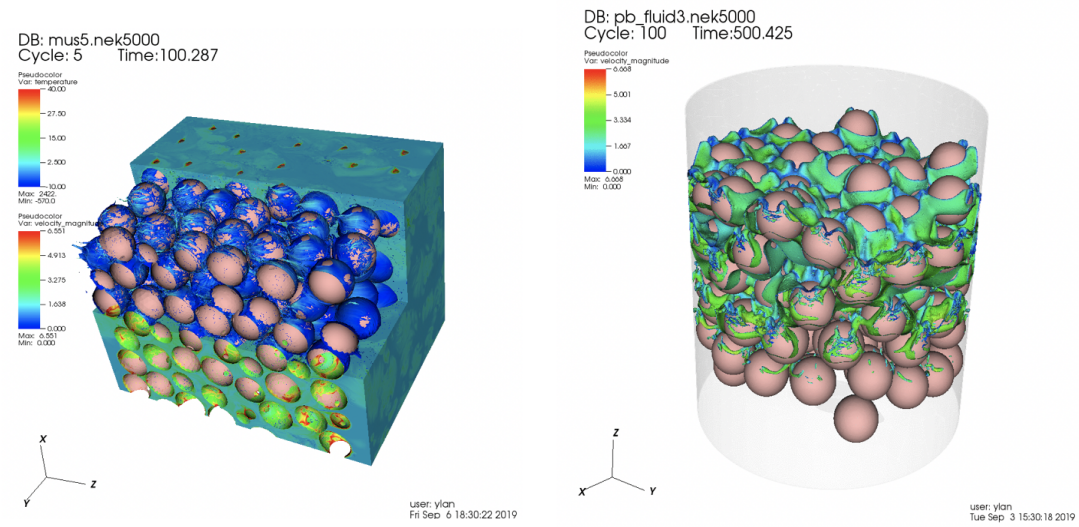

Figure 8: (Left) Temperature profile of pebble beds computed by Nek5000's steady solver based on JFNK method; 5 millions spectral element mesh, representing 363 pebbles ($E = 5M, N = 5$). (Right) Velocity profile of pebble beds computed by Nek5000 Navier-Stokes solver; $E = 365,000$ spectral element mesh, representing 148 pebbles.

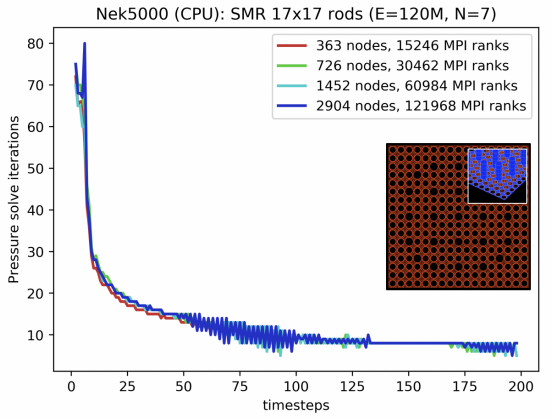

Figure 9: Nek5000 strong-scale, using 15246 to 121968 MPI ranks on Summit, for 17×17 rods with 120 millions of spectral elements, total 61 billions of gird points ($E = 120M, N = 7$).

reaches to steady state globally over time. The key ideas of the approach are how to compute the Jacobian matrix-vector product in Newton iterations and how to calculate the function $\underline{f}(\underline{u})$. We approximate the Jacobian matrix-vector product by the finite differences and $\underline{f}(\underline{u})$ obtained by the Nek5000's conventional BDF/extrapolation [5, 9] using a local timestep Δt. With proper choices between $\Delta\tau$ and Δt, our method allows a larger pseudo-transient timestep size $\Delta\tau^n$ as the global time-integration evolves and thus the total number of pseudo-transient time steps can be significantly reduced to reach to the steady solution.

Figure 8 (left) demonstrates the steady thermal flows obtained by Nek5000 JFNK solver while the velocity profile is given from pre-calculations. The geometry consists of 363 pebbles, simulated by 5 millions of spectral elements ($E = 5M$) with polynomial degree ($N = 5$). Our preliminary results show the JFNK is applicable for the complex geometry. The total pseudo-timestepping allows a very large timestep size (such

as $\Delta\tau = 10^{12}$) from the initial pseudo-timestep and reaches to steady state solution within 5 pseudo-time steps. However it currently requires 10∼20 Newton iterations at each pseudo-time step with large GMRES iteration counts in the Jacobian matrix-vector calculation process. Adding preconditioning to JFNK is a necessary component to speedup the computation and this is currently work in progress.

3.1.2 NekRS Challenges for Pebble Beds ($E = 365K, 5M, 60M$)

The pebble bed simulations are some of the largest ever undertaken with Nek5000/NekRS. As such, they stress several aspects of the Nek work flow, including meshing and partitioning. We have extended the recently-developed ParRSB parallel partitioner to support these very large cases. (Previous Nek5000 runs with 60 million elements used an ad-hoc preconditioner; with ParRSB we improved the Nek5000 runtimes by a factor of 4 on Mira.) Several of these extremely complex meshes have elements of rather poor quality and it has at this point required a fair amount of tuning to get things to run. The virtue of NekRS on Summit is that we can run problems of this scale with relative ease.

Our current efforts are ongoing for different sizes of problems representing 148∼ 6000 pebbles ($E = 365K \sim E = 60M$). Figure 8 (right) demonstrates Nek5000 run for the case of 148 pebbles with $E = 365K, N = 5$.

3.1.3 Nek5000 Strong-Scale for 17×17 *Rod Bundles* ($E = 120M$)

Figure 9 represents the pressure iterations for the largest case of 17×17 rod geometry run by Nek5000 with 120 millions of spectral elements, involving total grid points of 61 billions. We used Hypre AMG for the coarse grid solve, running with different number of MPI ranks for validation. This size of the problem was sensitive with the choice of coarsening strategies in Hypre. Current study is an important step to verify and validate NekRS at its development stage to move forward to larger size problems.

3.2 MARBL

MARBL is a next-gen multi-physics simulation code being developed at LLNL. One of the central features of MARBL is an ALE formulation based on the MFEM-driven BLAST code, which solves the conservation laws of mass, momentum, and energy using continuous and discontinuous Galerkin methods on a moving high-order mesh. The Lagrangian hydrodynamics phase [6, 7] of MARBL is based on high-order finite element discretization on a moving curved meshes, while MARBL's remap phase [3, 2] is based on a high-order discontinuous Galerkin finite element formulation.

The GPU port of MARBL/BLAST has been exclusively based on partial assembly technology from CEED and the GPU support via the MFEM version 4.0 release. This work has greatly benefited from the CEED-developed Laghos miniapp, which has been used for performance evaluation, vendor collaboration as well as procurement in the CORAL-2 and ATS-2.

During Q4 of FY19, we added partial assembly methods for momentum remap into BLAST; continued work on the GPU implementations of FCT remap and Lagrangian phase. Below we provide update on the performance of large-scale GPU of critical BLAST kernels in MFEM and Laghos.

In addition to these efforts, we collaborated with the MARBL team to complete an initial parallel version of Remhos, a miniapp for the Eulerian/Remap phase of BLAST, that includes computation of matrix-free low-order solution for DG transport/remap.

3.2.1 Large scale BP runs using MFEM

As reported in CEED-MS29, we introduced new GPU capabilities in MARBL through the MFEM version 4.0 release. In that report we showed initial results and performance comparison of the new GPU capabilities vs. single core and multicore-enabled backends, including single core vs. OpenMP raja-based and occa-based backends vs. MPI on single Skylake node with GV100 Nvidia GPU.

We have now extended and benchmarked these capabilities to multi-GPU settings as presented in Figure 10, where we illustrate the weak scalability for large scale BP3 runs on the Lassen supercomputer at LLNL using up to 1024 GPUs. This worked required specific kernels for MPI buffer packing/unpacking and while currently the GPU MPI communications go through the host, we are ready to take advantage of GPU-aware MPI.

The runs in Figure 10 use the libCEED-provide *CUDA-gen* backend from Section 2 which achieves over 2.5 GDOF/s performance on a single V100 GPU. While we observe a performance drop when going from serial to parallel (1 to 4 GPUs), the weak scalability from 4 to 1024 GPUs is quite good. At the largest end, we achieve a maximal performance of over 2 TDOF/s on problem sizes of over 30 billion degrees of freedom.

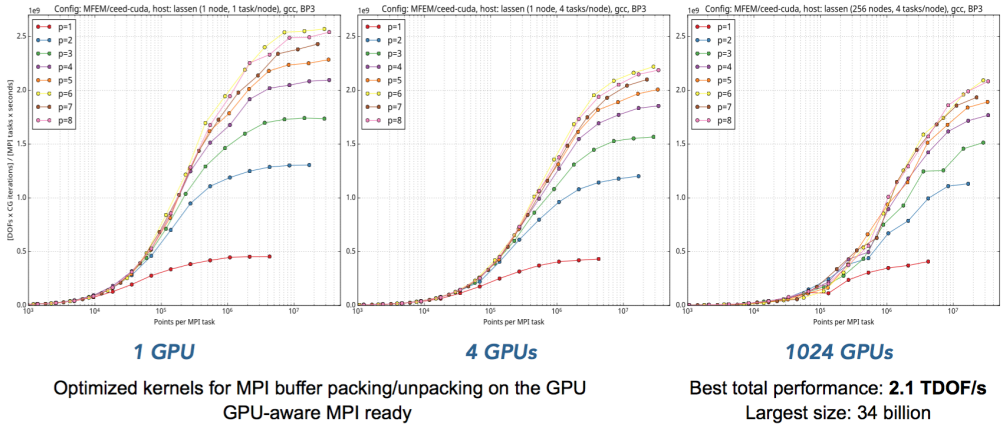

Figure 10: MFEM performance scalability on GPUs: MFEM-BP3, 3D, Lassen 4 x V100 GPUs / node.

3.2.2 Large scale Laghos runs using MFEM

The Laghos miniapp, is a simplified version of the BLAST Lagrange phase for the case of a single material and a simplified material stress. Nevertheless, the compute intensive, finite element operations for partial assembly of the mass and force matrices are identical, which allows the MARBL code to directly benefit from developments in optimizing / porting these kernels in the Laghos miniapp.

Laghos has followed all of the MFEM 4.0 latest developments and different test problems described in [[11]] have been added. As shown in the large scale BP runs using MFEM, similar runs were made on Lassen to solve these problems. All computations are performed on the GPU, everything is kept on the device, except for the result of the dot product that is brought back on to the CPU during the iterations of the CG solver.

Initial results are presented in figures 12 and 13. Figure 12 presents both the weak (gray lines) and strong (colored lines) scaling obtained on four to one thousands of GPUs during the CG (H1) iterations of the velocity solver, which corresponds to the BP2 CEED benchmark. Ideal strong scaling is possible for problem size large enough, while weak scaling is more easily reached through all the range of the runs. Figure 13 presents the throughput in dofs per second for the same solver, reaching 1.3 TDOF/s on 1024 GPUs. Figure 14 presents the throughput in dofs per second for the force kernel, reaching more than 4 TDOF/s on the same configuration.

3.3 ExaWind

The collaboration between CEED and ExaWind teams has focused on solving strategies for RANS models for large Reynolds number ($Re = 6M \sim 10M$) airfoil simulations. In previous milestones, we demonstrated results of wind turbine cross-section and NACA0012 airfoil. Our RANS approach was including the wall-function [12] and the wall-resolved models [14].

In this milestone, we continued investigating further improvement for the two different approaches for the boundary layer treatment in our Nek5000 RANS solver: (1) the wall-resolved *regularized* approach [14] where we have to use adequate resolution inside the very thin log and viscous sub-layers and (2) the new *wall-function* approach [12] where we do not need such high resolution as we move the boundary to a $y+$ of about 50-100. The wall function approach does not need to resolve the very sharp profiles immediately adjacent to the wall. Because of that, the source terms in the ω equation, which are treated explicitly, are not as large in the wall function approach and create less of a stability problem.

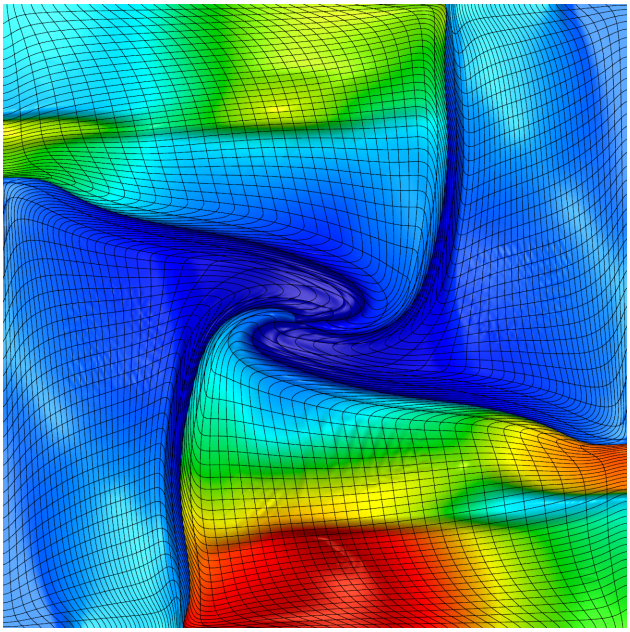

Figure 11: Laghos 2D problem number 6, as described in [11].

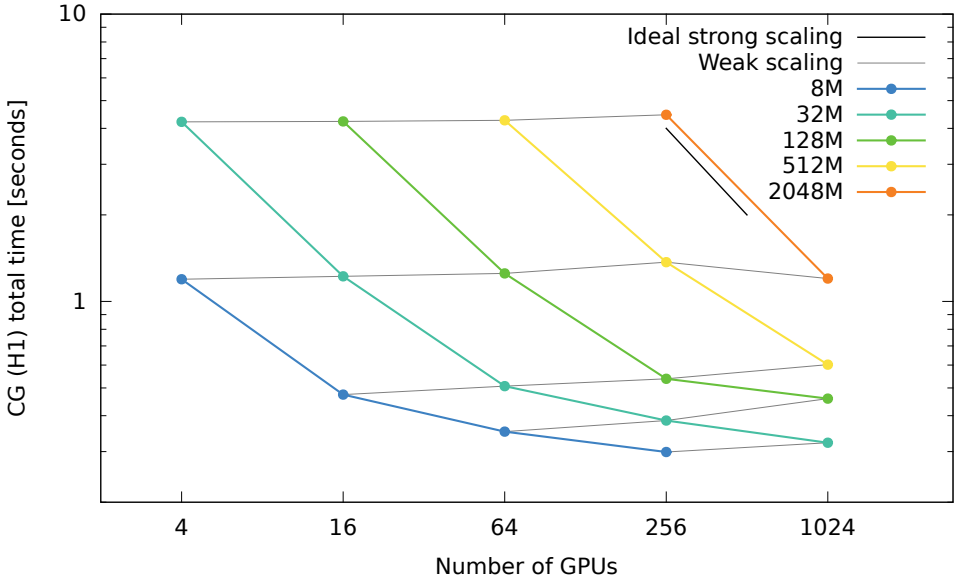

Figure 12: Initial weak and strong scaling results with Laghos and MFEM-4.0: 2D problem number 6 on Lassen, using up to 1024 GPUs. Ideal strong scaling is possible for problem size large enough, while weak scaling is reached through all the range of the runs.

Our RANS model is based on a regularized k-ω model, including the turbulent kinetic k and the specific dissipation rate ω in addition to the velocity field $\mathbf{v}$. The model describes the turbulent properties of incompressible flows with

$$k = \frac{\langle u'^2 \rangle + \langle v'^2 \rangle + \langle w'^2 \rangle}{2}, \tag{2}$$

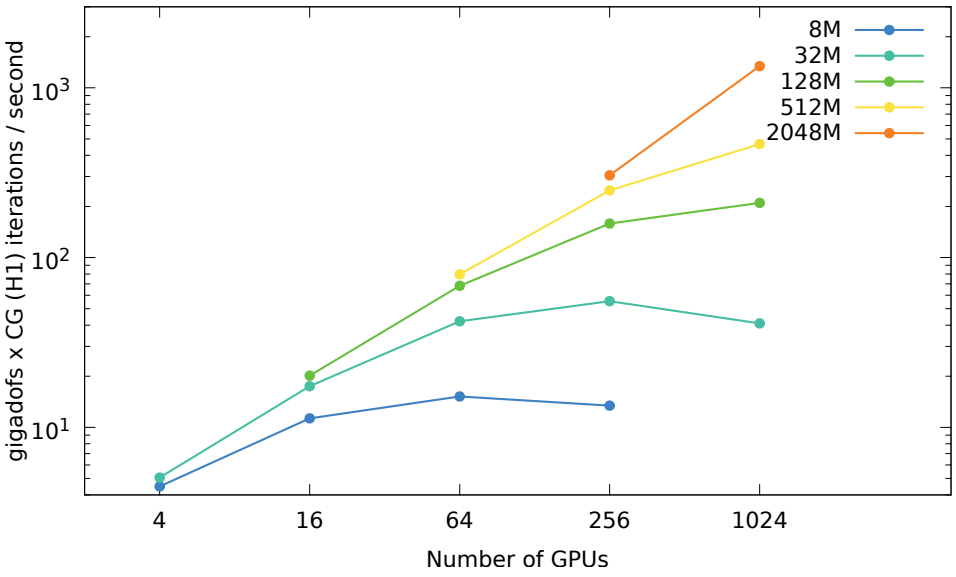

Figure 13: Throughput in GDOF/s for Laghos CG (H1) solver, reaching more than 1 TDOF/s on 1024 GPUs.

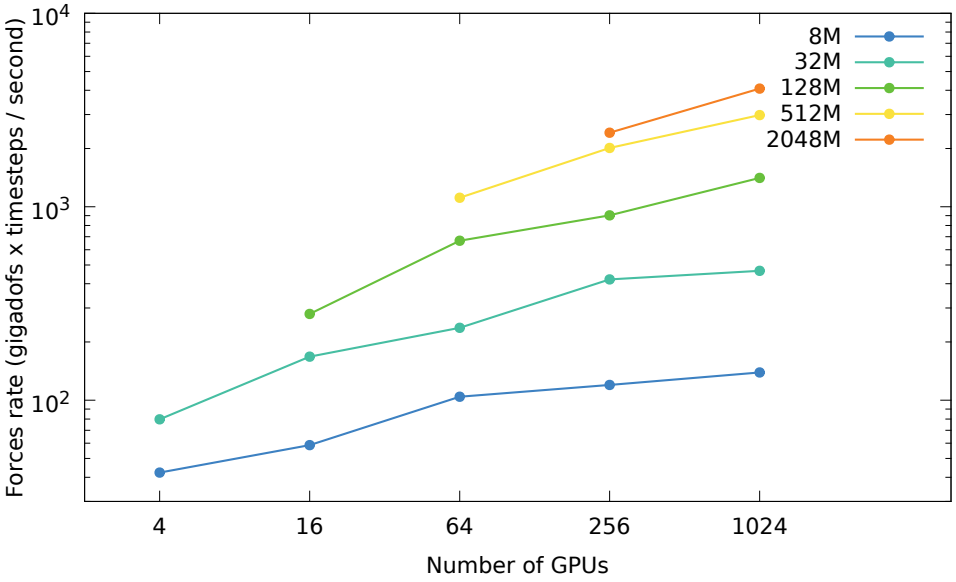

Figure 14: Throughput for the Laghos force kernel in (GDOF x timesteps / second), reaching more than 4 TDOF/s on 1024 GPUs.

where u', v' and w' are fluctuation component of velocity vector around the ensemble-averaged mean velocity

vector $\mathbf{v} = (u, v, w)$ governed by

$$\frac{\partial(\rho\mathbf{v})}{\partial t} + \nabla\cdot(\rho\mathbf{v}\mathbf{v}) = -\nabla p + \nabla\cdot\left[(\mu+\mu_t)\left(\nabla\mathbf{v} + \nabla\mathbf{v}^T - \frac{2}{3}\nabla\cdot\mathbf{v}\right)\right], \tag{3}$$

$$\frac{\partial(\rho k)}{\partial t} + \nabla\cdot(\rho k\mathbf{v}) = \nabla\cdot\left[(\mu+\frac{\mu_t}{\sigma_k})\nabla k\right] + P - \rho\beta^* k\omega, \tag{4}$$

$$\frac{\partial(\rho\omega)}{\partial t} + \nabla\cdot(\rho\omega\mathbf{v}) = \nabla\cdot\left[(\mu+\frac{\mu_t}{\sigma_\omega})\nabla\omega)\right] + \gamma\frac{\omega}{k}P - \rho\beta\omega^2 + S_\omega, \tag{5}$$

where μ is the molecular viscosity and μ_t is the turbulent viscosity with the continuity equation for incompressible flow

$$\nabla\cdot\mathbf{v} = 0, \tag{6}$$

and where S_ω is the cross-diffusion term which is non-zero only for the kw06. (We denote the Wilcox 1998 [16] and 2006 [17] versions as kw98 and kw06, respectively.) We have implemented and testing several RANS approaches in Nek5000, including the regularized $k-\omega$ model above, as well as some versions of the $k-\epsilon$ model and the $k-\tau$ model described below. We have tested these models in turbulent channel flow, flow past a backward facing step and external flows, such as flow past a wind turbine blade and the NACA0012 airfoil at 0 and 10 degrees angle of attack (aoa).

It was found that the $k-\tau$ model gives exactly the same results with the $k-\omega$ and for this reason we investigated this model more extensively and we plan to use it further in the future for the study of more complex flows. The main advantages of the $k-\tau$ model are 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.

Finally, as described in the next subsections, we have investigated ways to increase stability and robustness of our RANS approaches in order to be able to use a larger timestep. For this reason we have treated some of the terms appearing in the right-hand-side of the model equations implicitly for stabilization as explained below.

3.3.1 Stability Enhanced Wall-Resolved k-ω and k-τ Models

We have investigated the performance of several limiters, commonly used in the RANS literature, for the production terms in the k and ω equations as well as for the eddy viscosity. Two versions of the $k-\omega$ model were evaluated which differ in the values of some of the model coefficients; in addition the 2006 model includes a so-called cross-diffusion term S_ω.

It was found that for external flows it is important to limit the value of eddy viscosity in the far field which is typically not well resolved, in order to avoid abrupt variations in total viscosity. Such variations can cause stability problems to the simulation and/or can significantly increase the number of pressure/velocity iterations needed for convergence at every time-step. The performance of the modified version of the $k-\omega$ model was tested for flow past a NACA0012 airfoil geometry, with varying free stream conditions. With the modifications described above it was possible to obtain fully converted results for the drag and lift coefficients even for zero free stream values of k.

Tables 1 and 2 show the values of the drag and lift coefficient obtained using the two versions of the $k-\omega$ model (1998 and 2006) for flow past a NACA0012 airfoil at 0 and 10 degrees angle of attack (aoa). As can be observed, as resolution improves by increasing the polynomial order from $N = 7$ to $N = 11$ (8 and 12 points per direction, respectively), the model converges to the benchmark values of C_D and C_L which are also shown in the Table.

Another significant development was the implementation of the $k-\tau$ version of the $k-\omega$ model and the investigation of the scaling of all the terms appearing in the right-hand-side of the k and τ equations. The $k-\tau$ model was originally developed by Kalitzin et al. (1996, 2006) as an alternative implementation of the $k-\omega$ model. The equations for k and τ are derived from the $k-\omega$ equations using the definition $\tau = 1/\omega$:

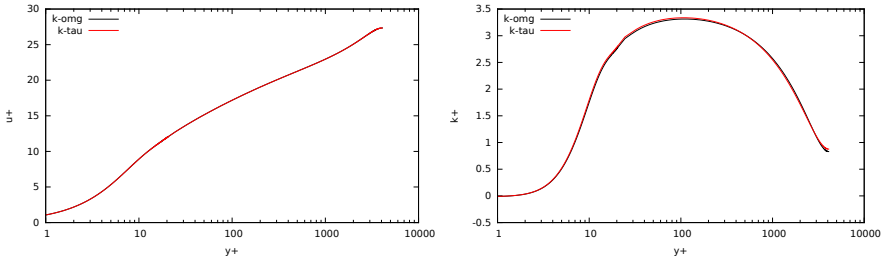

Figure 15: Comparison of mean streamwise velocity u^+ and turbulent kinetic energy k^+ profiles for $k-\omega$ and $k-\tau$ models.

$$\frac{\partial(\rho k)}{\partial t} + \nabla\cdot(\rho k\mathbf{v}) = \nabla\cdot\left[(\mu+\frac{\mu_t}{\sigma_k})\nabla k\right] + P - \rho\beta^*\frac{k}{\tau}, \tag{7}$$

$$\frac{\partial(\rho \tau)}{\partial t} + \nabla\cdot(\rho \tau\mathbf{v}) = \nabla\cdot\left[(\mu+\frac{\mu_t}{\sigma_\omega})\nabla \tau\right] - \gamma\frac{\tau}{k}P + \rho\beta - 2\frac{\mu}{\tau}\left(\nabla\tau\cdot\nabla\tau\right), \tag{8}$$

where now $G_k = P$, $Y_k = \beta^* k/\tau$, $G_\tau = \gamma\tau P/k$, $Y_\tau = \beta$, and $S_\tau = 2\nu\left(\nabla\tau\cdot\nabla\tau\right)/\tau$. In contrast to the original form of the $k-\omega$ model, in which the ω equation contains terms that become singular close to wall boundaries, all terms in the RHS of the k and τ equations reach a finite limit at walls and do not need to be treated asymptotically, i.e. they do not require regularization for numerical implementation. The last term in the τ equation was implemented in the form proposed by Kok and Spekreijse (2000) [10], which is the following:

$$S_\tau = 2\nu\left(\nabla\tau\cdot\nabla\tau\right)/\tau = 8\nu\left(\nabla\tau^{1/2}\cdot\nabla\tau^{1/2}\right) \tag{9}$$

Extensive verification tests of the $k-\tau$ model were performed for channel flow, flow past a backward facing step (BFS) and flow past the NACA0012 airfoil at aoa=0 and 10. Both versions of the model coefficients (kw98 and kw06) were implemented.

Figure 15 (left and right) show the comparison of the regularized $k-\omega$ and the $k-\tau$ model for flow in a channel at $Re = 10,950$. As can be observed the agreement of the two models is excellent for both the mean velocity as well as the kinetic energy.

The flow past a BFS at $Re = 149,700$ (Driver et al. 1990) was simulated using the $k-\tau$ model and results were compared with the results of the regularized $k-\omega$ model (Tomboulides et al. 2018). Figure 16, shows isocontours of mean streamwise velocity u and the k at steady state. The length of the recirculation zone from the RANS simulation using the $k-\tau$ model with the kw98 coefficients is equal to $6.59H$ (where H

Table 1: The drag (C_D) and lift (C_L) coefficients for aoa=0, the experimental data is from Ladson, NASA TM 4074, 1988.

	Nek5000 results				References		Exper.
Models	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 2: The drag (C_D) and lift (C_L) coefficients for aoa=10, the experimental data is from Ladson, NASA TM 4074, 1988.

	Nek5000 results				References		Experiment
Models	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

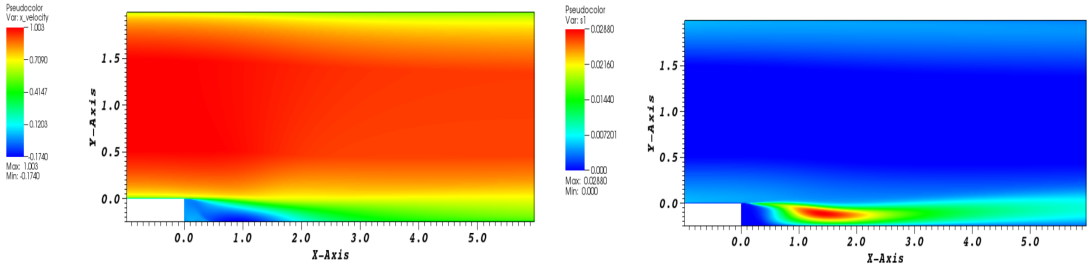

Figure 16: Comparison of mean streamwise velocity u^+ and turbulent kinetic energy k^+ profiles for the BFS at $Re = 149,700$ using the $k-\tau$ model of kw98.

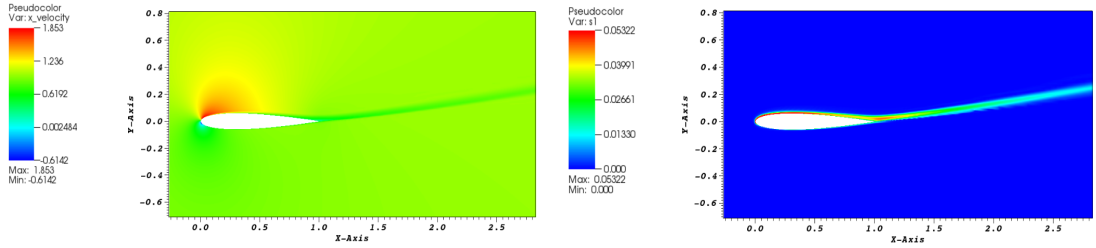

Figure 17: Comparison of mean streamwise velocity u^+ and turbulent kinetic energy k^+ profiles for the flow past a NACA0012 airfoil at $Re = 6 \times 10^6$ and a 10 degree angle of attack using the $k-\tau$ model of kw98.

is the step height) and was found to be in excellent agreement with the corresponding value obtained using the regularized $k-\omega$ model ($6.58H$). The recirculation zone length for the kw06 version is equal to $6.53H$.

We have also investigated the performance of the $k-\tau$ model and compared it with the regularized $k-\omega$ model results (described in the previous subsection) for the benchmark case of flow past the NACA0012 airfoil at aoa=0 and 10, which is relevant for external flows. The values of the drag and lift coefficients obtained with the $k-\tau$ model are also shown in Tables 1 and 2 for the two versions of the model using varying resolution at $Re = 6 \times 10^6$. As can be observed in these tables, the drag and lift coefficients are in very good agreement with both the regularized $k-\omega$ model results as well as with the benchmark values from the NASA LARC website. Isocontours of mean streamwise velocity u^+ and turbulent kinetic energy k^+ profiles at steady state for the case of $Re = 6 \times 10^6$ and aoa=10 and computed using the $k-\tau$ model of kw98 are shown in Figure 17.

3.3.2 Implicit Treatment of Source Terms in the Model Equations

A major development towards improving the robustness and efficiency of RANS in Nek5000 was the treatment of some of the source terms in the k and ω and the k and τ equations implicitly by including them in the left hand side of the equations. The terms that were treated implicitly are the following:

For the $k-\omega$ model (i) the dissipation term in the k equation (Y_k) and (ii) two terms in the right-hand-side of the ω equation which become unbounded close to walls at the same rate and need to be grouped together. For the $k-\tau$ model these terms are (i) the dissipation term in the k equation (Y_k) and (ii) the last term in the τ equation which can become a source of instability.
Treating all above terms implicitly did not increase the computational cost per timestep but allowed a significant increase of the timestep which is now only limited by the convective CFL condition.

3.3.3 Atmospheric Boundary Layer Benchmark Study

In collaboration with ExaWind team, we identified an atmospheric boundary layer benchmark problem [4], and tested out the case. The experimental setup uses the following parameters:

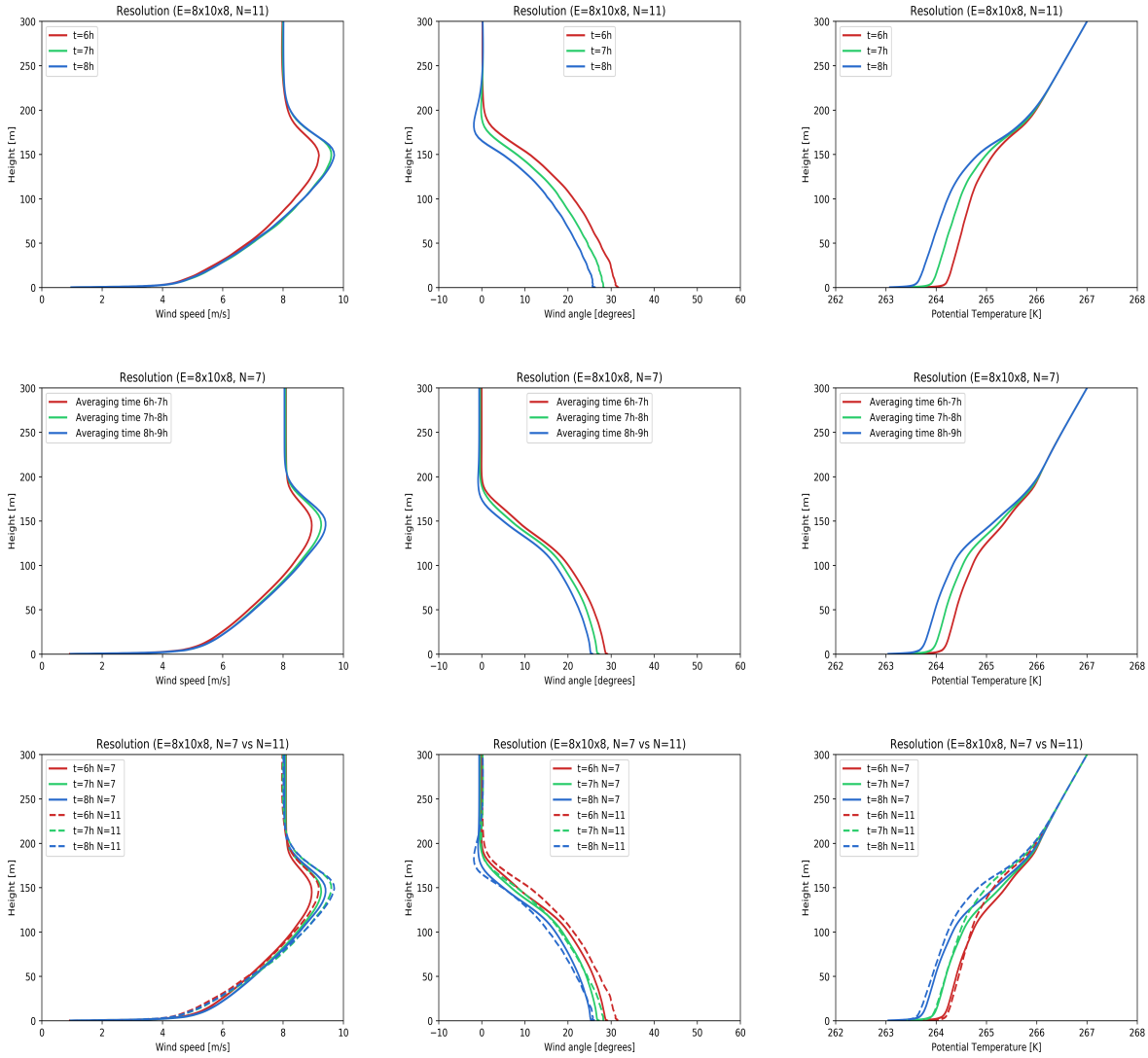

Figure 18: Comparison of wind speed, angle and potential temperature for varying N and time.

- Domain size: 400 m×400 m ×400 m
- Geostrophic wind speed in the x-direction of 8 m/s
- reference potential temperature of 263.5 K
- Resolution: 640 elements (8×8 in the two horizontal directions and 10 in the vertical direction). We used a polynomial order of $N = 7$ in each direction, corresponding to an effective resolution of 64^3. An additional simulation with polynomial order of 11 was also performed and both results are reported.
- initial conditions: constant x-velocity equal to geostrophic wind speed, potential temperature at the surface equal to 265 K up to 100 m and after that linearly increasing with rate 0.01 K/m.
- boundary conditions: Both no-slip as well as wall functions based on log-law were tried. Better agreement with benchmark was obtained using no-slip boundary condition at lower wall - with no need for additional resolution - and for this reason only these results are shown here.

- LES model: In these preliminary simulations we have used a relaxation LES model based on a high-pass filter (Schlatter et al.) which implemented through a relaxation term in the momentum equation. The use of more sophisticated physics-based LES models can be a topic of future work.

We performed Nek5000 simulations and verified that the computational setup was able to reproduce the Ekman analytical solution, demonstrated in Figure 18. Subsequently, the Re was set to $50,000$ and a numerical noise was added to the initial velocity and potential temperature profiles in order to enhance transition. After about 4–6 hours an Ekman layer started developing and a low level jet at a height of about 150 m started becoming apparent. The maximum jet wind speed observed was about 18–20% higher than the geostrophic wind speed.

The mean and rms wind velocity components and potential temperature were monitored for 9 hours and were horizontally averaged. The time-dependent behavior depends on the initial conditions and specifically the x- and z- components of the wind velocity do not necessarily reach their maximum values at the same time nor do they reach a quasi-steady state behavior concurrently. Several initial conditions were tried and the results here are only indicative.

3.4 ExaAM

The ECP Exascale Additive Manufacturing Application Project (ExaAM) team has been working on ExaConstit, a new application specifically for local property analysis, see Figure 19. This development is in collaboration with the CEED team. ExaConstit first release is focused on the ExaAM local property analysis, but as required, additional physics, inline results processing, and other features will be added to meet wider ExaAM needs. Significant integration between the ExaAM and CEED ECP activities is planned during this process. The application is currently found on a private repo of LLNLs Bitbucket and is in the process of being released.

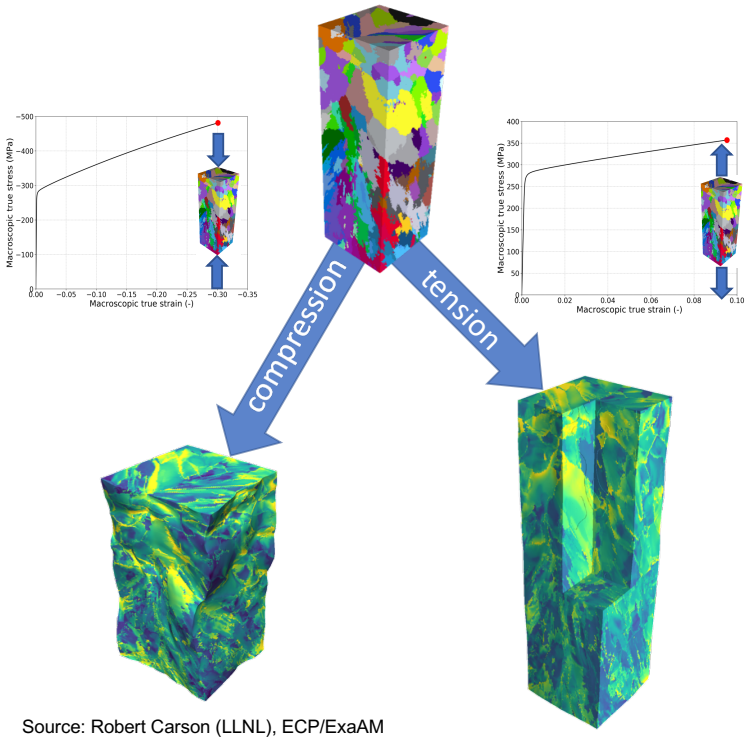

Figure 19: The ExaConstit miniapp aims to develop the capability to compute constitutive stress-strain curves given microstructural properties such as grain distribution and orientation.

Application development to date has focused on several different topics. Initial efforts were spent on

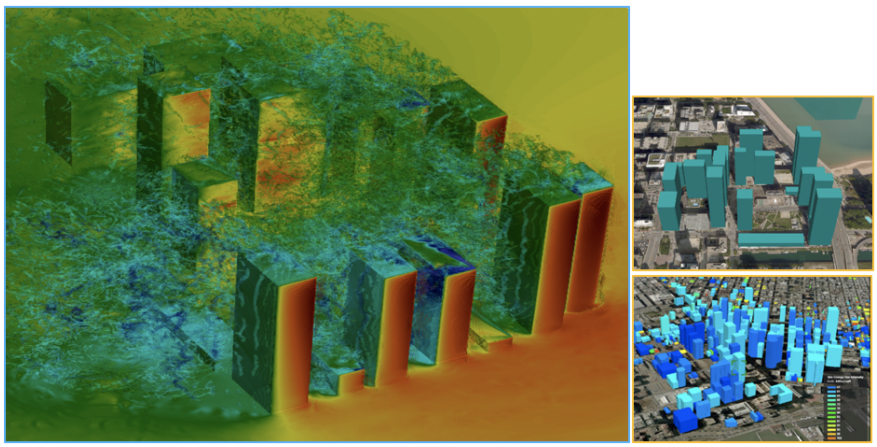

Figure 20: Chicago downtown Nek5000 LES simulations with WRF IC/BCs.

integrating the community-standard User-defined MATerial model (UMAT) constitutive model interface into the application code to run crystal-mechanics-based UMATs. Concurrent to this initial effort, an open source C++ GPU friendly crystal plasticity material modelling library, ExaCMech, was written to provide a viable path forward for porting ExaConstit over to the GPU. Engagement with the CEED team has begun on ways to reformulate ExaConstits global assembly operations over to a partial assembly formulation to take advantage of MFEMs new GPU capabilities that were introduced in the version 4.0 release. As a part of this effort, a miniapp is being developed for ExaCMech to examine the performance of different compute kernels and to engage with hardware vendors.

Several strong scaling studies of ExaConstit were conducted on Summit that made use of both material model interfaces (UMAT and ExaCMech). The timings from these runs serve as important points of reference for relevant aspects of the ExaAM challenge problem. Over the course of the ExaAM activity, we plan to make use of MFEMs capabilities for higher-order finite elements to achieve unprecedented fidelity in the resolution of microstructure-level material response features that are important for resolving initiation events such as the nucleation and growth of porosity.

3.5 Urban

In collaboration with the Urban team, the CEED team was able to perform Nek5000 simulations for the Chicago downtown block consisting of 20 buildings, demonstrated in Figure 20. This effort has increased the interest of NRC (Nuclear Regulatory Commission) for the quantification of 3D mixing effects resulting from new cooling tower and plant building wakes including SMR's plant where tighter footprint brings into question a validity of Gaussian plume models.

3.6 E3SM

The CEED team initiated a collaborative discussion with E3SM team, and began with running a simple case study of shallow water model using E3SM code, shown in Figure 21. We will further identify E3SM's atmospheric models related to the pressure gradient and hydrostatic balanced background state calculation that are needed at each time step. The CEED and E3SM teams are planning to work closely to setup the formulation for developing an efficient 2D elliptic solvers on spherical shells. CEED elliptic solvers (Nek5000, MFEM, libParanumal, and PETSc) would be very useful with extensive experience and expertise.

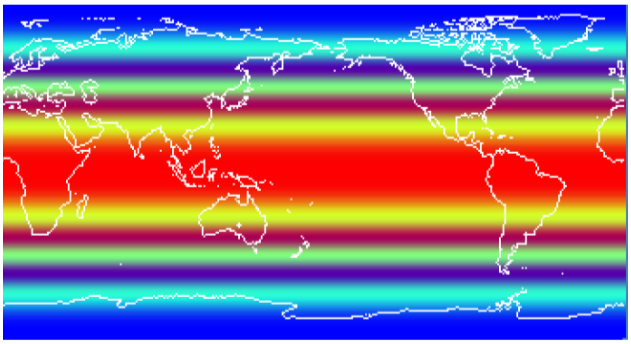

Figure 21: E3SM code: shallow water test example.

4. OTHER PROJECT ACTIVITIES

4.1 BP paper

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

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

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

This effort has been reported as a paper, *"Running Faster in HPC Applications"* by P. Fischer, M. Min, T. Rathnayake, S. Dutta, T. Kolev, V. Dobrev, J.-S. Camier, M. Kronbichler, T. Warburton, K. Swirydowicz, and J. Brown submitted for publication in the International Journal of High Performance Computing Applications [8].

4.2 CEED Third Annual Meeting

The third annual meeting of the CEED co-design center took place August 6–8 at Virginia Tech, Blacksburg, VA. Participants reported on the progress in the center, deepened existing and established new connections with ECP hardware vendors, ECP software technologies projects and other collaborators, planned project activities and brainstormed / worked as a group to make technical progress. In addition to gathering together many of the CEED researchers, the meeting included representatives of the ECP management, hardware vendors, software technology and other interested projects.

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

4.3 Initial NekRS release

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

4.4 Aurora Workshop at ANL

The CEED team participated the Aurora Programming (Sept. 17–19, 2019) held at ANL. The team successfully ported NekRS on the Aurora development system (Intel Gen9) during the workshop, and was able to get the baseline performance in comparison to NVIDIA V100 as shown in Table 3. Turbulent pipe flow is simulated for 100 time steps for the problem size of $E = 7480$ and $N = 7$.

4.5 Outreach

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

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

Table 3: NekRS baseline of performance measure on a single GPU, Intel Gen9 (Aurora development system) vs. Nvidia V100. Turbulent pipe simulations for 100 timestep runs with $E = 7840, N = 7$, total number of grid points with redundancy $n = 4,014,080$.

Systems	accumulated time, 100 steps	time per step	ratio (Gen9/V100)
Intel Gen9 (Iris@JLSE/ANL)	1.73837e+03 (sec)	17.38 (sec)	1
Nvidia V100 (Summit@OLCF)	4.42082e+01 (sec)	0.42 (sec)	41.38
Nvidia V100 (Nurburg@ANL)	4.14544e+01 (sec)	0.41 (sec)	42.39
Nvidia V100 (Monza@UIUC)	4.97899e+01 (sec)	0.49 (sec)	35.46

5. CONCLUSION

In this milestone, we developed architecture optimizations and tuning of performant discretization libraries and standards for finite element operators targeting heterogeneous systems. The focus was on delivering optimal data locality and motion, enhanced scalability and parallelism, derived through a number of CEED backends and software packages. We also delivered performance tuned CEED first and second-wave ECP applications.

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

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

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation's exascale computing imperative.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, LLNL-TR-792277.

REFERENCES

[1] A. Abdelfattah and S. Tomov (Organizers). Batched BLAS: API Standardization, Libraries, and Applications. `http://icl.utk.edu/bblas/siam-cse19/`, February 25 - March 1 2019. Mini-symposium at SIAM CSE'19, Spokane, Washington.

[2] 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.

[3] 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.

[4] M.J. Churchfield, Sang Lee, and P.J. Moriatry. Adding complex terrain and stable atmospheric condition capability to the OpenFOAM-based flow solver of the simulator for on/offshore wind farm application (SOWFA). *NREL/CP-5000-58539*, 2000.

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

[6] V. A. Dobrev, T. V. Kolev, and R. N. Rieben. High-order curvilinear finite element methods for Lagrangian hydrodynamics. *SIAM J. Sci. Comp.*, 34(5):B606–B641, 2012.

[7] V. A. Dobrev, T. V. Kolev, R. N. Rieben, and V. Z. Tomov. Multi-material closure model for high-order finite element Lagrangian hydrodynamics. *Internat. J. Numer. Methods Engrg.*, 82(10):689–706, 2016.

[8] P Fischer, M Min, T Rathnayake, S Dutta, T Kolev, V Dobrev, J.S. Camier, M Kronbichler, T Warburton, K Swirydowics, and J Brown. Running faster in hpc applications. *International Journal of High Performance Computing Applications*, page submitted, 2019.

[9] P. Fischer, M. Schmitt, and A. Tomboulides. *Recent developments in spectral element simulations of moving-domain problems*. Fields Institute Communications, Springer, 2017.

[10] J.C. Kok and S.P. Spekreijse. Efficient and accurate implementation of the k-ω turbulence model in the NLR multi-block Navier-Stokes system. *National Aerospace Laboratory NLR*, pages NLR–TP–2000–144, 2000.

[11] Alexander Kurganov and Eitan Tadmor. Solution of two-dimensional riemann problems for gas dynamics without riemann problem solvers. *Numerical Methods for Partial Differential Equations: An International Journal*, 18(5):584–608, 2002.

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

[13] Javier Martinez, Yu-Hsiang Lan, Elia Merzari, and Misun Min. On the use of LES-based turbulent thermal-stress models for rod bundle simulations. *Internaltional Journal of Heat and Mass Transfer*, 142:118399, 2019.

[14] A. Tomboulides, M. Aithal, P. Fischer, E. Merzari, A. Obabko, and D. Shaver. A novel numerical treatement 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.

[15] Ping-Hsuan Tsai, Yu-Hsiang Lan, Misun Min, and Paul Fischer. Jacobi-free Newton Krylov method for Poisson-Nernst-Planck equations. *to be submitted*, 2018.

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

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