ECP Milestone Report
Public release of CEED 2.0
WBS 2.2.6.06, Milestone CEED-MS25

Jed Brown
Ahmad Abdelfattah
Valeria Barra
Veselin Dobrev
Yohann Dudouit
Paul Fischer
Tzanio Kolev
David Medina
Misun Min
Thilina Ratnayaka
Cameron Smith
Jeremy Thompson
Stanimire Tomov
Vladimir Tomov
Tim Warburton

April 3, 2019

ECP Milestone Report
Public release of CEED 2.0
WBS 2.2.6.06, Milestone CEED-MS25

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

April 3, 2019

ECP Milestone Report
Public release of CEED 2.0
WBS 2.2.6.06, Milestone CEED-MS25

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	April 3, 2019	Original	

EXECUTIVE SUMMARY

In this milestone, we created and made publicly available the second full CEED software distribution, release CEED 2.0, consisting of software components such as MFEM, Nek5000, PETSc, MAGMA, OCCA, etc., treated as dependencies of CEED. The release consists of 12 integrated Spack packages for libCEED, mfem, nek5000, nekcem, laghos, nekbone, hpgmg, occa, magma, gslib, petsc and pumi plus the CEED *meta-package*.

The artifacts delivered include a consistent build system based on the above 13 Spack packages, documentation and verification of the build process, as well as improvements in the integration between different CEED components. As part of CEED 2.0, we also released the next version of libCEED (v0.4), which contains four new CPU backends, two new GPU backends, CPU backend optimizations, initial support for operator composition, performance benchmarking, and a Navier-Stokes demo. See the CEED website, `http://ceed.exascaleproject.org/ceed-2.0/` and the CEED GitHub organization, `http://github.com/ceed` for more details.

In addition to details and results from these efforts, in this document we report on other project-wide activities performed in Q2 of FY19, including: a version of the Laghos miniapp for CTS-2 procurement, extensive GPU performance baseline benchmarking for the Urban and ExaSMR applications, papers, minisimposium presentations, and other outreach efforts.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In this milestone, we created and made publicly available the second full CEED software distribution, release CEED 2.0, consisting of software components such as MFEM, Nek5000, PETSc, MAGMA, OCCA, etc., treated as dependencies of CEED. The release consists of 12 integrated Spack packages for libCEED, mfem, nek5000, nekcem, laghos, nekbone, hpgmg, occa, magma, gslib, petsc and pumi plus the CEED *meta-package*. This update included all new developments in the project since the CEED 1.0 release.

The artifacts delivered include a consistent build system based on the above 13 Spack packages, documentation and verification of the build process, as well as improvements in the integration between different CEED components. As part of CEED 2.0, we also released the next version of libCEED (v0.4), which contains four new CPU backends, two new GPU backends, CPU backend optimizations, initial support for operator composition, performance benchmarking, and a Navier-Stokes demo. See the CEED website, `http://ceed.exascaleproject.org/ceed-2.0/` and the CEED GitHub organization, `http://github.com/ceed` for more details.

2. CEED 2.0 PACKAGES

2.1 libCEED 0.4

libCEED version 0.4 was released in March 2019. The notable features of this release are new CPU and GPU backends, CPU backend optimizations, initial support for operator composition, performance benchmarking, and a Navier-Stokes demo.

2.1.1 New libCEED Backends

Six new backends are included in this release, four CPU backends and two GPU backends.

The new CPU backends come in two families. The `/cpu/self/*/serial` backends process one element at a time and are intended for meshes with a smaller number of high order elements. The `/cpu/self/*/blocked` backends process blocked batches of eight interlaced elements and are intended for meshes with higher numbers of elements. The `/cpu/self/avx/*` backends rely upon AVX instructions to provide vectorized CPU performance. The `/cpu/self/xsmm/*` backends rely upon the LIBXSMM package to provide vectorized CPU performance.

The `/gpu/cuda/*` backends provide GPU performance strictly using CUDA. The `/gpu/cuda/ref` backend is a reference CUDA backend, providing reasonable performance for most problem configurations. The `/gpu/cuda/reg` backend uses a simple parallelization approach, where each thread treats a finite element. Using just in time compilation, provided by nvrtc (NVidia Runtime Compiler), and runtime parameters, this backend unroll loops and map memory address to registers. The `/gpu/cuda/reg` backend achieve good peak performance for 1D, 2D, and low order 3D problems, but performance deteriorates very quickly when threads run out of registers.

2.1.2 Operator Composition

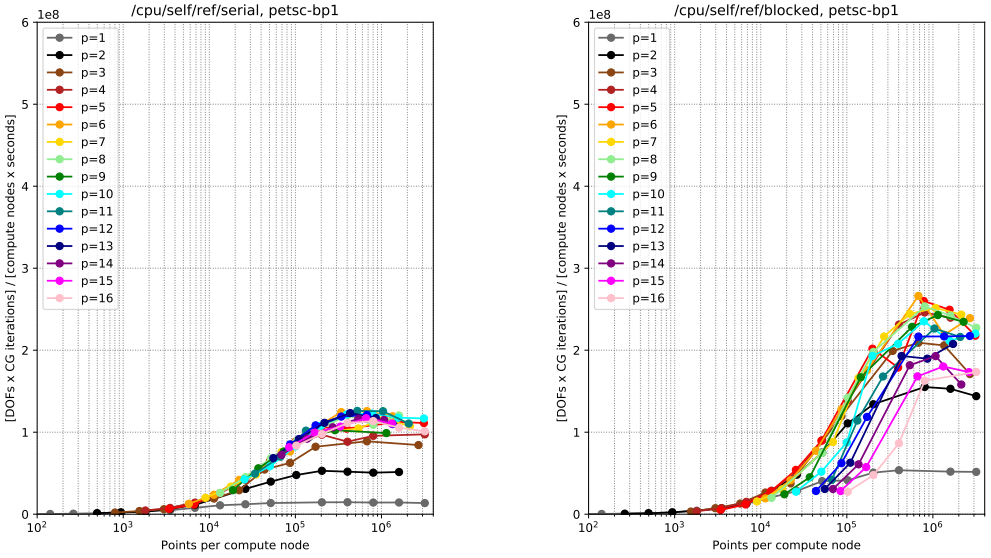
Initial aupport for composing libCEED operators was added in version 0.4. The new capability supports heterogeneous elements, such as mixed mesh or multi-physics problems. Operator composition is currently only supported by CPU backends. Future releases will extend operator composition to GPU backends and optimize performance.

2.1.3 Benchmarking libCEED Performance

Performance benchmarking was added using PETSc implementations of Benchmark Problems 1 and 3. This benchmarking can be used to fine tune the build flags for a particular environment and determine the most performant backend.

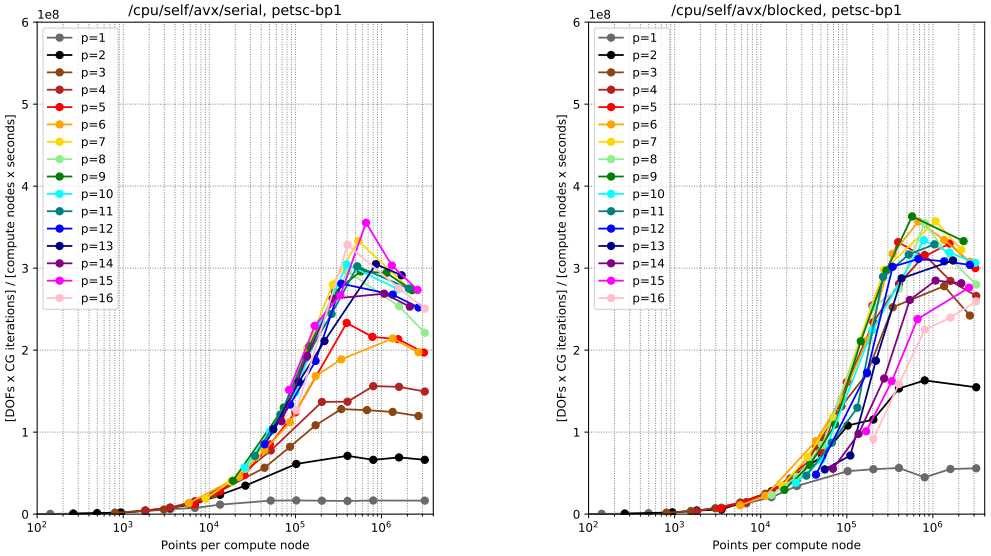
The following benchmark plots were generated on University of Colorado Boulder's Summit machine using four full nodes with Intel Xeon E5-2680 v3 processors. Five benchmarking runs were averaged to reduce the

noise in the results. The first six plots show the results of Benchmark Problem 1 using reference backends in pure C and optimized backends using AVX instructions and LIBXSMM. The plots on the left show the backends using internal vectorization for each element and the plots on the right show the backends using external vectorization across blocks of eight elements.



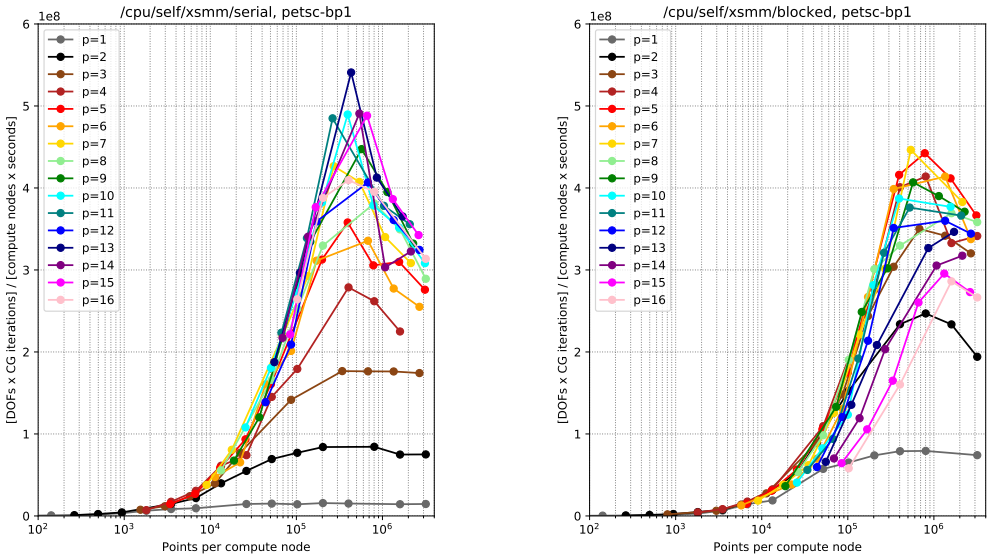
(a) Internal Vectorization **(b)** External Vectorization

Figure 1: Pure C Refrence Backends



(a) Internal Vectorization **(b)** External Vectorization

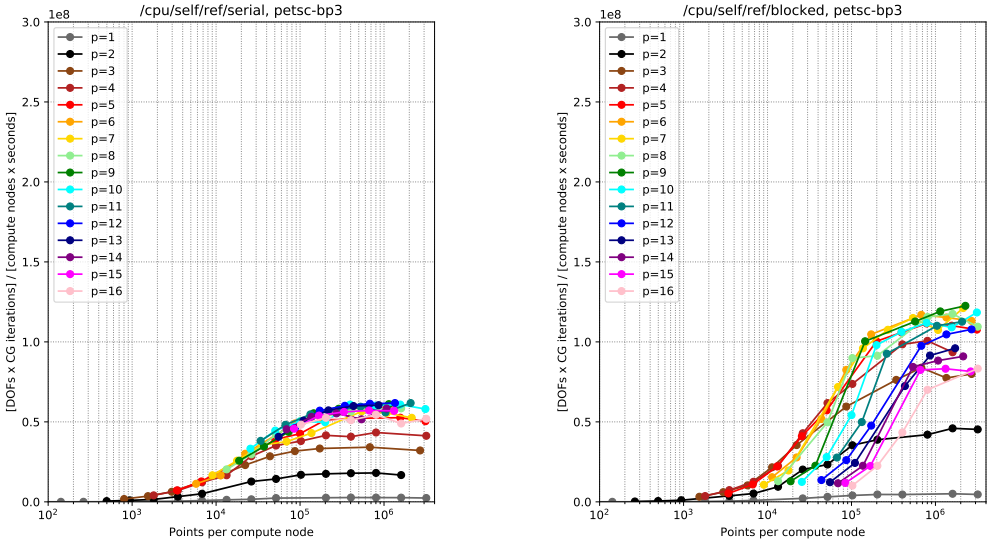
Figure 2: AVX Backends



(a) Internal Vectorization **(b)** External Vectorization

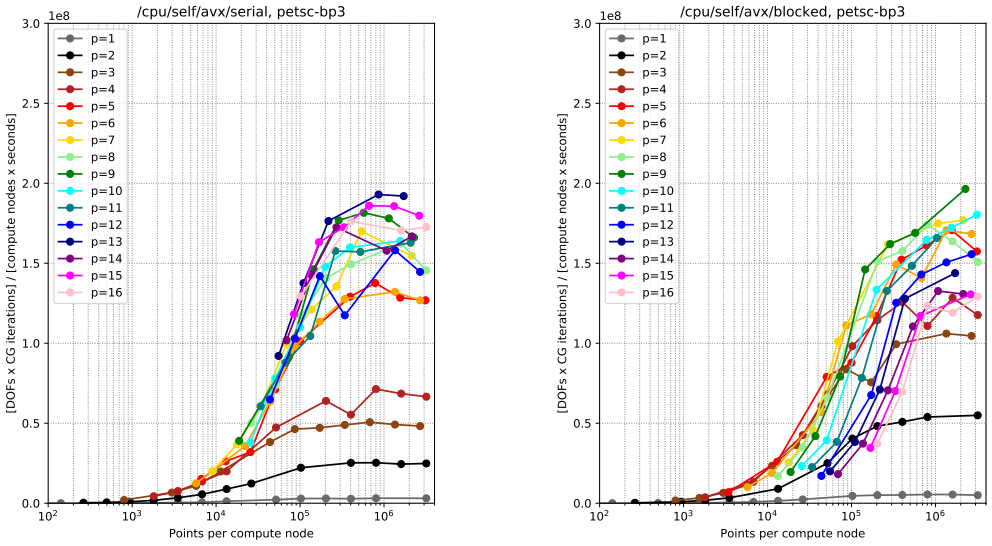
Figure 3: LIBXSMM Backends

The next six plots show the results of Benchmark Problem 3 for the same six backends described above.



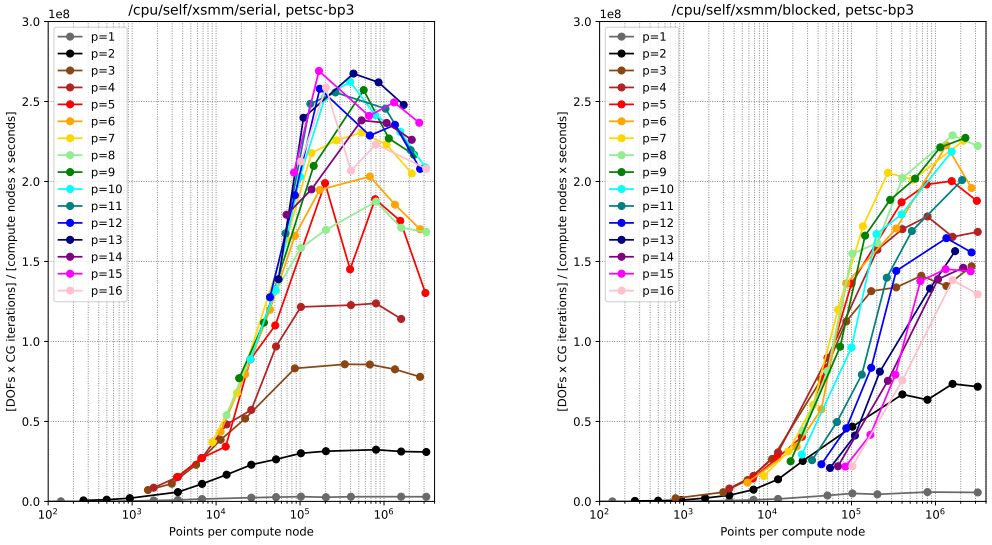
(a) Internal Vectorization **(b)** External Vectorization

Figure 4: Pure C Refrence Backends



(a) Internal Vectorization **(b)** External Vectorization

Figure 5: AVX Backends



(a) Internal Vectorization **(b)** External Vectorization

Figure 6: LIBXSMM Backends

2.1.4 Navier-Stokes example

A new explicit Navier-Stokes example was added to the family of libCEED examples. This example represents work being done towards a miniapp for a Navier-Stokes solver, that fully exploits libCEED capabilities. The goal of the libCEED team was to provide users with an example that is more involved in terms of the physics solved for (therefore, also in terms of the corresponding mathematical description and `QFunction` implementation), compared to the existing CEED Bake-off Problems (BPs) (see `https://ceed.exascaleproject.org/bps/`), so that more complex problems of interest to the scientific community

can be efficiently solved by using libCEED.

This example solves the time-dependent Navier-Stokes equations of compressible gas dynamics in a static Eulerian three-dimensional frame using structured high-order finite element/spectral element spatial discretizations and explicit high-order time-stepping. The Navier-Stokes example has been developed using PETSc, so that the pointwise physics (defined at quadrature points) is separated from the parallelization and meshing concerns.

The mathematical formulation is given in what follows. The compressible Navier-Stokes equations in conservative form are

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \boldsymbol{U} = 0\,, \tag{1a}$$

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \left(\frac{\boldsymbol{U} \otimes \boldsymbol{U}}{\rho} + P \mathbf{I}_3 \right) + \rho g \hat{\boldsymbol{k}} = \nabla \cdot \boldsymbol{\sigma}\,, \tag{1b}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot \left(\frac{(E+P)\boldsymbol{U}}{\rho} \right) = \nabla \cdot \left(\boldsymbol{u} \cdot \boldsymbol{\sigma} + k \nabla T \right)\,, \tag{1c}$$

where $\boldsymbol{\sigma} = \mu(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T + \lambda(\nabla \cdot \boldsymbol{u})\mathbf{I}_3)$ is the Cauchy (symmetric) stress tensor, with μ the dynamic viscosity coefficient, and $\lambda = -2/3$ the Stokes hypothesis constant. In equations (1), ρ represents the volume mass density, U the momentum density (defined as $\boldsymbol{U} = \rho \boldsymbol{u}$, where $\boldsymbol{u}$ is the vector velocity field), E the total energy density (defined as $E = \rho e$, where e is the total energy), $\mathbf{I}_3$ represents the 3×3 identity matrix, g the gravitational acceleration constant, $\hat{\boldsymbol{k}}$ the unit vector in the z direction, k the thermal conductivity constant, T represents the temperature, and P the pressure, given by the following equation of state

$$P = (c_p/c_v - 1)\left(E - \boldsymbol{U} \cdot \boldsymbol{U}/(2\rho) - \rho g z\right)\,, \tag{2}$$

where c_p is the specific heat at constant pressure and c_v is the specific heat at constant volume (that define $\gamma = c_p/c_v$, the specific heat ratio).

The system (1) can be rewritten in vector form

$$\frac{\partial \boldsymbol{q}}{\partial t} + \nabla \cdot \boldsymbol{F}(\boldsymbol{q}) = S(\boldsymbol{q})\,, \tag{3}$$

for the state variables 5-dimensional vector

$$\boldsymbol{q} = \begin{pmatrix} \rho \\ \boldsymbol{U} \equiv \rho \mathbf{u} \\ E \equiv \rho e \end{pmatrix} \begin{matrix} \leftarrow \text{ volume mass density} \\ \leftarrow \text{ momentum density} \\ \leftarrow \text{ energy density} \end{matrix} \tag{4}$$

where the flux and the source terms, respectively, are given by

$$\boldsymbol{F}(\boldsymbol{q}) = \begin{pmatrix} \boldsymbol{U} \\ (\boldsymbol{U} \otimes \boldsymbol{U})/\rho + P\mathbf{I}_3 - \boldsymbol{\sigma} \\ (E+P)\boldsymbol{U}/\rho - (\boldsymbol{u} \cdot \boldsymbol{\sigma} + k\nabla T) \end{pmatrix}\,, \tag{5a}$$

$$S(\boldsymbol{q}) = -\begin{pmatrix} 0 \\ \rho g \hat{\boldsymbol{k}} \\ 0 \end{pmatrix}. \tag{5b}$$

For the spatial discretization, we use high-order finite elements/spectral elements, namely, the high-order Lagrange polynomials defined over non-uniformly spaced nodes, the Legendre-Gauss-Lobatto (LGL) points (roots of the p^{th}-order Legendre polynomial P_p). We discretize the domain, $\Omega \subset \mathbb{R}^3$, by letting $\Omega = \bigcup_{e=1}^{N_e} \Omega_e$, with N_e disjoint hexahedral elements. The physical coordinates are $\mathbf{x} = (x, y, z) \in \Omega_e$, while the reference coordinates are $\boldsymbol{\xi} = (\xi, \eta, \zeta) \in \mathbf{I} = [-1, 1]^3$. Let the discrete solution be

$$\mathbf{q}_N(\boldsymbol{x}, t)^{(e)} = \sum_{k=1}^{P} \psi_k(\boldsymbol{x}) \boldsymbol{q}_k^{(e)} \tag{6}$$

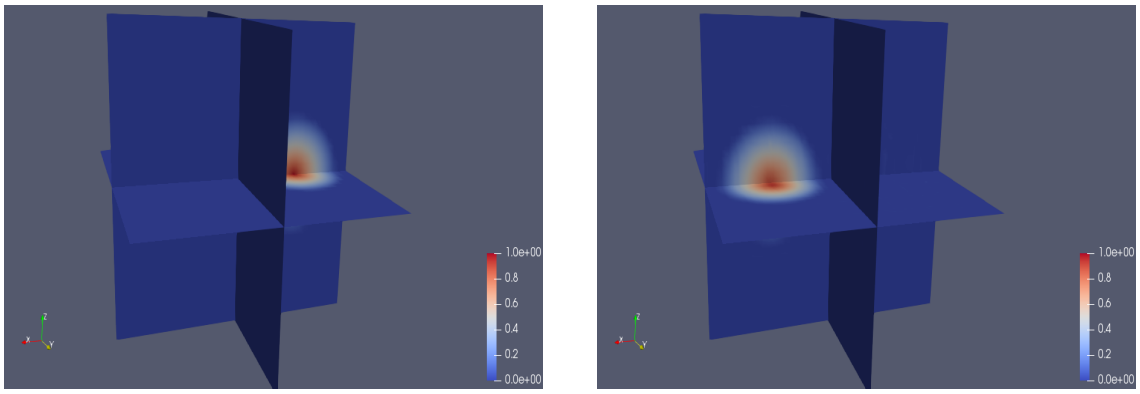


Figure 7: Advection problem. A blob of energy is advected by a uniform circular flow, with constant speed $|\mathbf{u}| = 0.5$ m/s. On the left, the initial condition at $t = 0$ s; on the right, the blob has been transported for half a circle, at $t = 12$ s. This simulation has been run with degree 6 polynomials, $Q = 8$ number of quadrature points along one dimension, 512 elements of $[0, 500]^3$ m dimension.

with P the number of nodes in the element (e). We use tensor-product bases $\psi_{kji} = h_i(\xi)h_j(\eta)h_k(\zeta)$.

For the time discretization, we use an explicit formulation

$$\frac{\boldsymbol{q}_N^{n+1} - \boldsymbol{q}_N^n}{\Delta t} = -[\nabla \cdot \boldsymbol{F}(\boldsymbol{q}_N)]^n + [S(\boldsymbol{q}_N)]^n \,, \tag{7}$$

solved with the adaptive Runge-Kutta-Fehlberg (RKF4-5) method by default (any explicit time-stepping scheme avaialble in PETSc can be chosen at runtime).

The Navier-Stokes solver can be built by moving from the CEED main directory

```
cd libCEED/examples/navier-stokes
```

and by running

```
make
```

For testing and validation purposes the libCEED team has identified a set of problems for the Navier-Stokes solver (see `https://github.com/CEED/libCEED/tree/master/examples/navier-stokes`).

The Advection problem. A simplified version of system (1), only accounting for the transport of total energy, is given by

$$\frac{\partial E}{\partial t} + \nabla \cdot (\boldsymbol{u}E) = 0 \,, \tag{8}$$

with $\boldsymbol{u}$ the vector velocity field. In this particular test case, a blob of total energy (defined by a characteristic radius r_c) is transported by a uniform circular velocity field. We have solved (8) with no-slip and non-penetration boundary conditions for $\boldsymbol{u}$, and no-flux for E. This problem can be run with

```
./navierstokes -problem advection
```

The size of the domain can be specified by the command line options `lx, ly, lz`, the element resolution with `resx, resy, resz`, and the characteristic radius of the energy blob by `rc`. For instance, a domain $\Omega = [0, 3000]^3$ m can be discretized with elements of 1000^3 m size, with

```
./navierstokes -problem advection -lx 3000 -ly 3000 -lz 3000 -resx 1000 -resy
1000 -resz 1000 -rc 800
```

See Figure 7 for an example of results obtained for this problem type.

The Density Current problem. For this test problem, a cold air bubble (of radius r_c) drops by convection in a neutrally stratified atmosphere. Its initial condition is defined in terms of the Exner pressure, $\pi(\boldsymbol{x}, t)$,

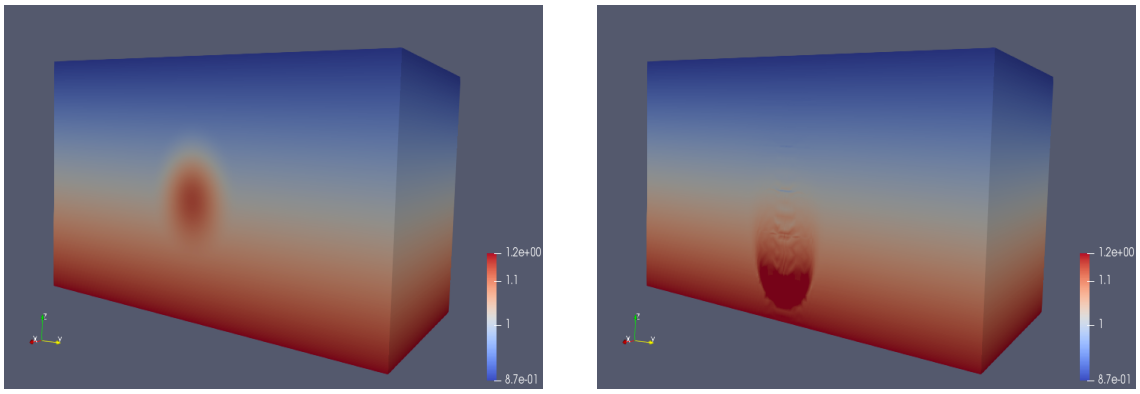


Figure 8: Density current problem. A bubble of cold air in the stratified atmosphere falls under its own weight and is transported towards the ground by convective currents. In this figure we display the volume density of the bubble. On the left, the initial condition at $t = 0$ s; on the right, the blob has dropped at $t = 73$ s. This simulation has been run with degree 9 polynomials, $Q = 13$ number of quadrature points along one dimension, 864 elements of $[0, 1000]^3$ m dimension.

and potential temperature, $\theta(\boldsymbol{x}, t)$, that relate to the state variables via

$$\rho = \frac{P_0}{(c_p - c_v)\theta(\boldsymbol{x}, t)} \pi(\boldsymbol{x}, t)^{\frac{c_v}{c_p - c_v}} , \tag{9a}$$

$$e = c_v \theta(\boldsymbol{x}, t)\pi(\boldsymbol{x}, t) + \boldsymbol{u} \cdot \boldsymbol{u}/2 + gz , \tag{9b}$$

where P_0 is the atmospheric pressure. For this problem, we have used no-slip and non-penetration boundary conditions for $\boldsymbol{u}$, and no-flux for mass and energy densities. This problem can be run with

```
./navierstokes -problem density_current
```

Similarly to the advection example, the user can specify the size of the physical domain, the element resolution, and the characteristic radius of the thermal bubble, together with all other physical parameters involved, at runtime. See Figure 8 for an example of results obtained for this problem type.

For both problems, the user can specify how often to print outputs (in terms of number of steps), by adding the command line option `-output_freq`. For instance, to indicate to print outputs every 1000 steps

```
./navierstokes -output_freq 1000
```

By doing so, outputs for the solution in `vts` format will be generated. Moreover, the libCEED team has developed checkpoints by printing the last calculated solution and the time stamp at which this solution was printed in binary format, so that the user can subsequently continue an existing simulation via the command line option `-continue`, followed by the step number of the solution checkpoint that is desired to be continued. For instance, to continue a simulation from the output number 2000

```
./navierstokes -continue 2000
```

For a list of all available options see `https://github.com/CEED/libCEED/tree/master/examples/navier-stokes/README.md`.

2.2 MFEM 3.4

The CEED 2.0 distribution includes version 3.4 of MFEM, which adds the following new features compared to CEED 1.0:

- More general and efficient mesh adaptivity

  - Added support for PUMI, the Parallel Unstructured Mesh Infrastructure from `https://scorec.rpi.edu/pumi`. PUMI is an unstructured, distributed mesh data management system that is capable of handling general non-manifold models and effectively supports automated adaptive analysis. PUMI enables for the first time support for parallel unstructured modifications of MFEM meshes.
  - Significantly reduced MPI communication in the construction of the parallel prolongation matrix in `ParFiniteElementSpace`, for much improved parallel scaling of non-conforming AMR on hundreds of thousands of MPI tasks. The memory footprint of the `ParNCMesh` class has also been reduced.
  - In `FiniteElementSpace`, the fully assembled refinement matrix is now replaced by default by a specialized refinement operator. The operator option is both faster and more memory efficient than using the fully assembled matrix. The old approach is still available and can be enabled, if needed, using the new method `FiniteElementSpace::SetUpdateOperatorType()`.

- Discretization improvements
  - Added support for a general *high-order*-to-*low-order refined* transfer of `GridFunction` and true-dof data from a *high-order* finite element space defined on a coarse mesh, to a *low-order refined* space defined on a refined mesh. The new methods, `GetTransferOperator` and `GetTrueTransferOperator` in the `FiniteElementSpace` classes, work in both serial and parallel and support matrix-based as well as matrix-free transfer operator representations. They use a new method, `GetTransferMatrix`, in the `FiniteElement` class similar to `GetLocalInterpolation`, that allows the coarse `FiniteElement` to be different from the fine `FiniteElement`.
  - Added class `ComplexOperator`, that implements the action of a complex operator through the equivalent 2×2 real formulation. Both symmetric and antisymmetric block structures are supported.
  - Added classes for general block nonlinear finite element operators (deriving from `BlockNonlinearForm` and `ParBlockNonlinearForm`) enabling solution of nonlinear systems with multiple unknowns in different function spaces. Such operators have assemble-based action and also support assembly of the gradient operator to enable inversion with Newton iteration.
  - Added variable order NURBS: for each space each knot vector in the mesh can have a different order. The order information is now part of the finite element space header in the NURBS mesh output, so NURBS meshes in the old format need to be updated.
  - In the classes `NonlinearForm` and `ParNonlinearForm`, added support for non-conforming AMR meshes.
  - New specialized time integrators: symplectic integrators of orders 1-4 for systems of first order ODEs derived from a Hamiltonian and generalized-alpha ODE solver for the filtered NavierStokes equations with stabilization. See classes `SIASolver` and `GeneralizedAlphaSolver` in `linalg/ode.hpp`.
  - Inherit finite element classes from the new base class `TensorBasisElement`, whenever the basis can be represented by a tensor product of 1D bases.
  - Added support for elimination of boundary conditions in block matrices.

- New and updated examples and miniapps
  - Added a new serial and parallel example (`ex19`) that solves the quasi-static incompressible hyperelastic equations. The example demonstrates the use of block nonlinear forms as well as custom block preconditioners.
  - Added a new electromagnetics miniapp, Maxwell, for simulating time-domain electromagnetics phenomena as a coupled first order system of equations.
  - A simple local refinement option has been added to the mesh-explorer miniapp (menu option `'r'`, sub-option `'l'`) that selects elements for refinement based on their spatial location - see the function `region()` in the source file.

  - Added a set of miniapps specifically focused on Isogeometric Analysis (IGA) on NURBS meshes in the `miniapps/nurbs` directory. Currently the directory contains variable order NURBS versions of examples 1, 1p and 11p.
  - Added PUMI versions of examples `ex1`, `ex1p`, `ex2` and `ex6p` in a new `examples/pumi` directory. The new examples demonstrate the PUMI APIs for parallel and serial mesh loading (`ex1` and `ex1p`), applying BCs using classification (`ex2`), and performing parallel mesh adaptation (`ex6p`).
  - Added two new miniapps related to `DataCollection` I/O in `miniapps/tools`: `load-dc.cpp` can be used to visualize fields saved via `DataCollection` classes; `convert-dc.cpp` demonstrates how to convert between MFEM's different concrete `DataCollection` options.
  - Example 10p with its SUNDIALS and PETSc versions have been updated to reflect the change in the behavior of the method `ParNonlinearForm::GetLocalGradient()` and now works correctly on non-conforming AMR meshes. Example 10 and its SUNDIALS version have also been updated to support non-conforming AMR meshes.

- Miscellaneous
  - Documented project workflow and provided contribution guidelines in the new top-level file, `CONTRIBUTING.md`.
  - Added (optional) Conduit Mesh Blueprint support of MFEM data for both in-core and I/O use cases. This includes a new `ConduitDataCollection` that provides json, simple binary, and HDF5-based I/O. Support requires Conduit $\geq$ v0.3.1 and VisIt $\geq$ v2.13.1 will read the new Data Collection outputs.
  - Added a new developer tool, `config/sample-runs.sh`, that extracts the sample runs from all examples and miniapps and runs them. Optionally, it can save the output from the execution to files, allowing comparison between different versions and builds of the library.
  - Support for building a shared version of the MFEM library with GNU make.
  - Added a build option, `MFEM_USE_EXCEPTIONS=YES`, to throw an exception instead of calling `abort` on MFEM errors.
  - When building with the GnuTLS library, switch to using X.509 certificates for secure socket authentication. Support for the previously used OpenPGP keys has been deprecated in GnuTLS 3.5.x and removed in 3.6.0. For secure communication with the visualization tool GLVis, a new set of certificates can be generated using the latest version of the script `glvis-keygen.sh` from GLVis.
  - Upgraded MFEM to support Axom 0.2.8. Prior versions are no longer supported.

2.3 Nek5000 version 19.0-rc2

The CEED 2.0 distribution includes version 19.0-rc2 of Nek5000 adds the following new features compared to CEED 1.0:

- Enhanced scalability
  - Uncoupled multi-session (`neknek`) simulations
  - Gather scatter operations across sessions
  - Gather scatter options across gtp-planes
  - `par` file support for postnek
  - `mkSIZE` to automatically create SIZE file
  - Object and boundary handling
  - Support `ParMETIS` partitioner
- Enhanced algorithmic supports
  - RANS k-ω and k-ω-SST (experimental)

  - Online mesh-smoother (experimental)
  - `FEM_AMG` preconditioner (experimental) `p40=3`.
  - `SEMG_AMG_HYPRE` preconditioner (experimental) `p40=2`.
  - Lagrangian phase model - LPM (experimental)

- Miscellaneous
  - ElapsedTime option for writeControl (in par)
  - Print runtime-statistics every 100 steps
  - Support for GNU 8.x compilers
  - Support for Cray compilers
  - Support for ARM compilers
  - `CHT` support for generic fld reader
  - Overwrite core routines in usr

2.4 PETSc 3.11

PETSc 3.11 includes contributions from nearly 90 contributors since v3.8.3 (included in CEED-1.0). Some notable features for CEED include

- `VecScatter` support for use of shared memory within a node and for using `PetscSF` (heavily used for unstructured mesh operations),
- AVX-512 optimizations and new matrix formats (sliced ELLPACK and MKL inspector-executor),
- higher performance scalable sparse matrix-matrix products and $P^T AP$ operations (used by algebraic multigrid setup),
- improved parallel mesh IO and hierarchical operations,
- pipelined Krylov methods with improved stability properties, and
- Python-3.4+ support (in addition to Python-2.6+).

2.5 OCCA 1.0.8

OCCA has released versions 1.0.0 through 1.0.8 thanks to 6 contributors, which include the following features and updates

- OKL parser improvements
- Enabled kernel compilation using AMD's HIP
- Added kernel type safety by giving `occa::memory` (C++) and `occaMemory` (C) optional type hints
- Test coverage improvement (57.9% → 70.9%)
- Enabled OKL `@attributes` through `#pragma occa attributes` which can be used in regular C / C++ code
- Released initial Python API which supports building OKL kernels through Python's native function syntax

2.6 PUMI 2.2.0

The principal feature of PUMI 2.2.0 is inclusion in the December 2018 XSDK-0.4.0 release. Several additional developments are listed below.

- Travis CI nightly testing (3450a945)
- User documentation of mesh and geometric model support (`https://github.com/SCOREC/core/wiki/Geometric-Models`, `https://github.com/SCOREC/core/wiki/Mesh-Generation`)
- Field cloning (6f3c3926)
- Generalizations of field accumulation (d783b19e)
- Compatibility with Simmetrix SimModSuite 12.0.180811 (06df5a0f)
- Support for discontinuous Galerkin methods with Simmetrix meshes (1727b32b)

2.7 MAGMA 2.5.0

The CEED 2.0 distribution includes developments from the MAGMA 2.4 release (on 06/25/18) and the latest MAGMA 2.5 release (on 01/02/2019). This includes routines for batched computations and support for the fast FP16 Tensor Cores arithmetic in Nvidia V100 GPUs, namely:

- Batched device interface routines for use in tensor contractions;
- Performance improvements across many batch routines, including batched TRSM, batched LU, batched LU-nopiv, and batched Cholesky;
- Batched GEMM with support for strided access of the matrices;
- GEMM in FP16 arithmetic (HGEMM) as well as auxiliary functions to cast matrices from FP32 to FP16 storage (magmablas_slag2h) and from FP16 to FP32 (magmablas_hlag2s);
- Mixed-precision solver that is able to provide an FP64 solution with up to 4X speedup using the fast FP16 Tensor Cores arithmetic in the Nvidia V100 GPUs.

2.8 Laghos 2.0

The CEED 2.0 distribution includes version 2.0 of Laghos, which introduces four new variants of the miniapp, namely,

- pure CUDA version, where the major computation kernels are rewritten in CUDA without utilizing any intermediate abstraction layers;
- OCCA version, based on the OCCA abstraction layer;
- RAJA version, based on the RAJA abstraction layer;
- AMR version, demonstrating dynamic adaptive mesh refinement for moving curved meshes of arbitrary order.

2.9 HPGMG 0.4

HPGMG 0.4 includes several improvements to the FV and FE variants, including

- AVX-512 support for FE,
- CEED BP solver configurations and extra quadrature points for FE,
- removal of FE process grid restriction for larger prime factors, and
- layout flexibility and padding/alignment support for FV.

2.10 gslib v.1.0.4

gslib v1.0.4 includes major features and improvements:

- Added support for `gs_float` in Fortran `gs`.
- Updated testing unit with new header files.

2.11 NekCEM

Newly supported features of NekCEM include the following:

- Multi-session run support for large-scale parameter studies that represent structural differences between optical devices. This has been tested up to millions of parameter studies for phase analysis of metals.
- Structural support for OLCF/Summit runs; Measure performance on Summit, in comparison to Titan. (See Section 4.2)

3. CEED 2.0 TESTING AND DISTRIBUTION

The CEED 2.0 distribution consists of 12 Spack packages (plus a CEED meta-package), documented at `https://ceed.exascaleproject.org/ceed-2.0/`. In this section we review the configuration settings for several common and leadership computing architectures.

3.1 Mac

The configuration file below, also included as `ceed2-darwin-highsierra-x86_64-packages.yaml` in the CEED 2.0 distribution, provides a sample `packages.yaml` file based on Homebrew, that should work on most Macs. (You can use MacPorts instead of Homebrew if you prefer.)

```
packages:
    all:
        compiler: [clang]
        providers:
            blas: [veclibfort]
            lapack: [veclibfort]
            mpi: [openmpi]
    openmpi:
        paths:
            openmpi@3.0.0: ~/brew
        buildable: False

    cmake:
        paths:
            cmake@3.10.2: ~/brew
        buildable: False
    cuda:
        paths:
            cuda@9.1.85: /usr/local/cuda
        buildable: False
    libx11:
        paths:
            libx11@system: /opt/X11
        version: [system]
        buildable: False
    libxt:
        paths:
            libxt@system: /opt/X11
        version: [system]
        buildable: False
    xproto:
        paths:
            # see /opt/X11/lib/pkgconfig/xproto.pc
            xproto@7.0.31: /opt/X11
```

```
        version: [7.0.31]
        buildable: False
    python:
        paths:
            python@2.7.10: /usr
        buildable: False
    zlib:
        paths:
            zlib@1.2.11: /usr
        buildable: False
```

The packages in `/brew` were installed with `brew install package`. If you don't have Homebrew, you can install it and the needed tools with:

```
git clone https://github.com/Homebrew/brew.git
cd brew
bin/brew install openmpi cmake python zlib
```

The packages in `/usr` are provided by Apple and come pre-built with Mac OS X. The `cuda` package is provided by NVIDIA and should be installed separately by downloading it from NVIDIA. We are using the Clang compiler, Open MPI, and Apple's BLAS/LAPACK accelerator library.

3.2 Linux RHEL7

The configuration file below, also included as `ceed2-linux-rhel7-x86_64-packages.yaml` in the CEED 2.0 distribution, provides a sample 'packages.yaml' file that can be adapted to work on a RHEL7 Linux desktop

```
packages:
    all:
        compiler: [gcc]
        providers:
            mpi: [openmpi]
            blas: [netlib-lapack]
            lapack: [netlib-lapack]
    netlib-lapack:
        paths:
            netlib-lapack@system: /usr/lib64
        buildable: False
    openmpi:
        paths:
            openmpi@3.0.0: ~/local
        buildable: False

    cmake:
        paths:
            cmake@3.10.2: ~/local
        buildable: False
    cuda:
        paths:
            cuda@9.1.85: ~/local/cuda
        buildable: False
    libx11:
        paths:
            libx11@system: /usr
        version: [system]
        buildable: False
    libxt:
        paths:
            libxt@system: /usr
        version: [system]
        buildable: False
    xproto:
        paths:
            xproto@7.0.32: /usr
        version: [7.0.32]
        buildable: False
    python:
```

```
        paths:
             python@2.7.14: /usr
        buildable: False
    zlib:
        paths:
            zlib@1.2.11: /usr/lib64
        buildable: False
```

The above file uses user-installed Open MPI, CMake and CUDA packages, with the rest of the CEED prerequisites installed via the *yum* package manager.

3.3 Linux Ubuntu

A very similar file, `ceed2-ubuntu18.10-packages.yaml` provides Spack configuration for the Ubuntu distribution:

```
packages:
    all:
        compiler: [gcc]
        providers:
            mpi: [mpich]
            blas: [openblas]
            lapack: [openblas]
    openblas:
        paths:
            openblas@system: /usr/lib
        buildable: False
    mpich:
        paths:
            mpich@3.3: /usr/local
        buildable: False

    cmake:
        paths:
            cmake@3.12.1: /usr
        buildable: False
    libx11:
        paths:
            libx11@system: /usr
        version: [system]
        buildable: False
    libxt:
        paths:
            libxt@system: /usr
        version: [system]
        buildable: False
    xproto:
        paths: # See /usr/share/pkgconfig/xproto.pc for version
            xproto@7.0.32: /usr
        buildable: False
    python:
        paths:
             python@3.6.7: /usr
        buildable: False
    zlib:
        paths:
            zlib@1.2.11: /usr/lib
        buildable: False
```

In this case we use GCC and other development packages via `apt install` and with MPICH installed separately (as needed to use containerized HPC environments like Shifter and Singularity). You can use `docker pull jedbrown/ceed-base` to get a build environment that is ready for `spack install ceed`.

3.4 Cori

The CEED2.0 installation on Cori (NERSC) via the Spack package manager can be obtained by downloading the Spack configuration file for Cori, that can be found on the CEED website `https://ceed.exascaleproject.org/ceed-2.0/`.

After downloading the file, it needs to be renamed from `ceed2-cori-packages.yaml` to `packages.yaml` and placed in the folder `/.spack/` or `/.spack/<platform>/` on Cori. This file contains information on the packages that the CEED2.0 installation requires, so that the user does not need to install them from scratch, which can take a long time. The Spack configuration file for Cori is the following

```
packages:
    all:
        compiler: [gcc@7.3.0, intel@18.0.5.274]
        providers:
            mpi: [mpich]
            mkl: [intel-mkl]
            blas: [intel-mkl, cray-libsci]
            scalapack: [intel-mkl, cray-libsci]
            pkgconfig: [pkg-config]
    mpich:
        modules:
            mpich@3.2%gcc@7.3.0 arch=cray-cnl9-haswell: cray-mpich
            mpich@3.2%intel@18.0.5.274 arch=cray-cnl9-haswell: cray-mpich
        buildable: False
    intel-mkl:
        buildable: false
        paths:
            intel-mkl@2018.3.222%intel: /opt/intel
            intel-mkl@2018.3.222%gcc: /opt/intel
    pkg-config:
        buildable: false
        paths:
            pkg-config@0.28: /usr
    cmake:
        modules:
            cmake@3.14.0%gcc@7.3.0 arch=cray-cnl9-haswell: cmake
            cmake@3.14.0%intel@18.0.5.274 arch=cray-cnl9-haswell: cmake
        buildable: False
    libx11:
        paths:
            libx11@system: /usr
        version: [system]
        buildable: False
    libxt:
        paths:
            libxt@system: /usr
        version: [system]
        buildable: False
    xproto:
        paths: # See /usr/lib64/pkgconfig/xproto.pc for version
            xproto@7.0.28: /usr
        buildable: False
    python:
        paths:
            python@2.7.13: /usr
        buildable: False
    boost:
        modules:
            boost@1.69.0%gcc@7.3.0 arch=cray-cnl9-haswell: boost
            boost@1.69.0%intel@18.0.5.274 arch=cray-cnl9-haswell: boost
        buildable: False
    m4:
        modules:
            m4@1.4.17%gcc@7.3.0 arch=cray-cnl9-haswell: m4
            m4@1.4.17%intel@18.0.5.274 arch=cray-cnl9-haswell: m4
        buildable: False
    openssl:
        modules:
            openssl@1.1.0a%gcc@7.3.0 arch=cray-cnl9-haswell: openssl
            openssl@1.1.0a%intel@18.0.5.274 arch=cray-cnl9-haswell: openssl
        buildable: False
    perl:
        paths:
```

```
            perl@5.18.2%gcc@7.3.0 arch=cray-cnl9-haswell: /usr
            perl@5.18.2%intel@18.0.5.274 arch=cray-cnl9-haswell: /usr
        buildable: False
    autoconf:
        modules:
            autoconf@2.69%gcc@7.3.0 arch=cray-cnl9-haswell: autoconf
            autoconf@2.69%intel@18.0.5.274 arch=cray-cnl9-haswell: autoconf
        buildable: False
    automake:
        modules:
            automake@1.15%gcc@7.3.0 arch=cray-cnl9-haswell: automake
            automake@1.15%intel@18.0.5.274 arch=cray-cnl9-haswell: automake
        buildable: False
```

3.5 ALCF Theta

The configuration file below, also included as `ceed2-theta-packages.yaml` in the CEED 2.0 distribution, provides a sample 'packages.yaml' for the Theta system at ALCF.

```
packages:
  cmake:
    paths:
      cmake@3.5.2%gcc@8.2.0 arch=cray-CNL-mic_knl: /usr
      cmake@3.5.2%intel@16.0.3.210 arch=cray-CNL-mic_knl: /usr
    buildable: False
  python:
    paths:
      python@2.7.13%gcc@8.2.0 arch=cray-CNL-mic_knl: /usr
      python@2.7.13%intel@16.0.3.210 arch=cray-CNL-mic_knl: /usr
    buildable: False
  pkg-config:
    paths:
      pkg-config@0.28%gcc@8.2.0 arch=cray-CNL-mic_knl: /usr
      pkg-config@0.28%intel@16.0.3.210 arch=cray-CNL-mic_knl: /usr
    buildable: False
  autoconf:
    paths:
      autoconf@2.69%gcc@8.2.0 arch=cray-CNL-mic_knl: /usr
      autoconf@2.69%intel@16.0.3.210 arch=cray-CNL-mic_knl: /usr
    buildable: False
  automake:
    paths:
      automake@1.13.4%gcc@8.2.0 arch=cray-CNL-mic_knl: /usr
      automake@1.13.4%intel@16.0.3.210 arch=cray-CNL-mic_knl: /usr
    buildable: False
  libtool:
    paths:
      libtool@2.4.2%gcc@8.2.0 arch=cray-CNL-mic_knl: /usr
      libtool@2.4.2%intel@16.0.3.210 arch=cray-CNL-mic_knl: /usr
    buildable: False
  m4:
    paths:
      m4@1.4.16%gcc@8.2.0 arch=cray-CNL-mic_knl: /usr
      m4@1.4.16%intel@16.0.3.210 arch=cray-CNL-mic_knl: /usr
    buildable: False
  intel-mkl:
    paths:
      intel-mkl@16.0.3.210%intel@16.0.3.210 arch=cray-CNL-mic_knl: /opt/intel
    buildable: False
  mpich:
    modules:
      # requires 'module load cce' otherwise gives parsing error
      mpich@7.6.3%gcc@8.2.0 arch=cray-CNL-mic_knl: cray-mpich/7.6.3
      mpich@7.6.3%intel@16.0.3.210 arch=cray-CNL-mic_knl: cray-mpich/7.6.3
    buildable: False
  boost:
    paths:
```

```
      boost@1.64.0%gcc@8.2.0 arch=cray-CNL-mic_knl: /soft/libraries/boost/1.64.0/gnu
      boost@1.64.0%intel@16.0.3.210 arch=cray-CNL-mic_knl: /soft/libraries/boost/1.64.0/
    intel
    buildable: False
  all:
    providers:
      mpi: [mpich]
    compiler: [gcc@8.2.0]
```

3.6 Other machines: OLCF Summit, LLNL TOSS3 and Lassen

For Spack configurations tested on other machines, please consult the CEED 2.0 release website `https://ceed.exascaleproject.org/ceed-2.0`.

4. GPU PERFORMANCE BASELINES FOR CEED APPLICATIONS

In responding to the needs of ECP Urban and ExaSMR teams, we measured baseline performance of our current versions of Nek and libParanumal codes on Summit, in comparison to Titan.

4.1 Urban: libParanumal performance on Summit vs. Titan

A number of performance improvements were recently implemented in libParanumal's incompressible Navier-Stokes flow solver, including improved JIT compilation of OCCA for 1000s of MPI ranks, experiments with overlapping additive Schwarz and communication avoiding, pipelined preconditioned conjugate gradient solvers. With these changes, the libParanumal solver matched the NekBone miniapp within 10% in CPU mode. In GPU mode, the libParanumal solver demonstrated weak GPU scaling at approximately 90% on 1/2 of both Summit and Titan.

An experimental version (NekRS) has been under development that is based on integration of Nek5000 + libParanumal. As a part of collaboration with ECP Urban team, we measured the speedup of NekRS on Summit (vs. Titan) for incompressible Navier-Stokes solver for Taylor-Green vortex simulations shown in Figure 9 and demonstrated the baseline performance in Figure 10.

Our testing setups for baseline performance on Summit and Titan are summarized in Table 1. We performed simulations for 1000 timesteps with 8 iterations fixed for velocity and pressure. We measured maximum size per GPU, $n = EN^3$ with $N = 7$, keeping the ratio 2.666 (8000/3000) which is equivalent to the ratio of the memory size per GPU, 16GB on Summit and 6GB on Titan. While the total #GPUs are 27648 on Summit and 18688 on Titan, we keep the same ratio with 1.479. We achieve 4.92× speedup in simulation time and 7.2× speedup in DOFs/sec per step as shown in Figure 10. From the ratio of the memory bandwidth (900GB/s on Summit and 250GB/s on Titan), the speedup in timing would be ~ 5.3 and the theoretical speedup would be 8.8x (5.3×1.5) on Summit with 1.5× more resources.

Table 1: Experimental code NekRS run for 1000 timesteps with 8 iterations fixed for velocity and pressure. The ratio of data size 2.666 (8000/3000). Performance is shown in Figure 10.

system	E per GPU	N	$n = EN^3$	$DOFs = 4 \times n$ per GPU
Summit	8000	7	2.7 M	10.8 M
Titan	3000	7	1.0 M	4 M

4.2 Urban: NekCEM performance on Summit vs. Titan

As a part of collaboration with ECP Urban team, we also measured the speedup of NekCEM on Summit (vs. Titan). NekCEM's GPU implementation is based on OpenACC and the code is fully explicit solver which is highly optimized. We use the similar experimental setup and demonstrate the baseline performance for periodic traveling wave simulations shown in Figure 11. Our testing setups for baseline performance on Summit and Titan are summarized in Table 2. We performed 1000 timesteps, involving stage 4th-order

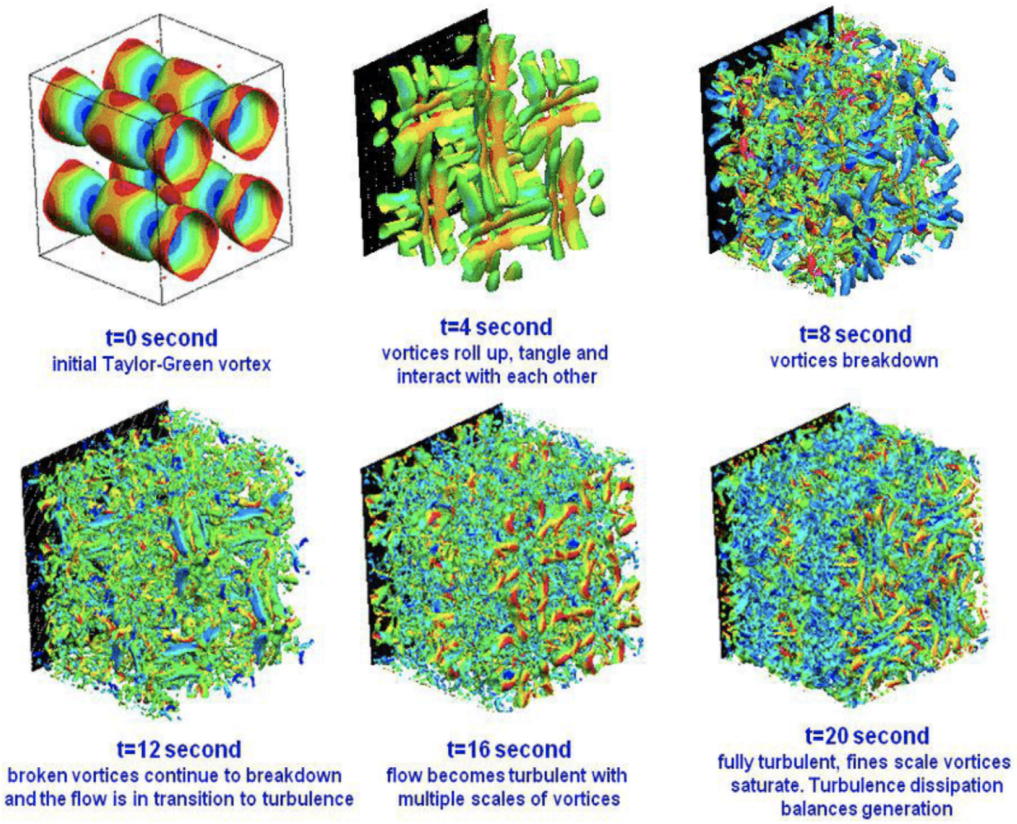


Figure 9: Nek5000+libParanumal (NekRS): Taylor-Green vortex.

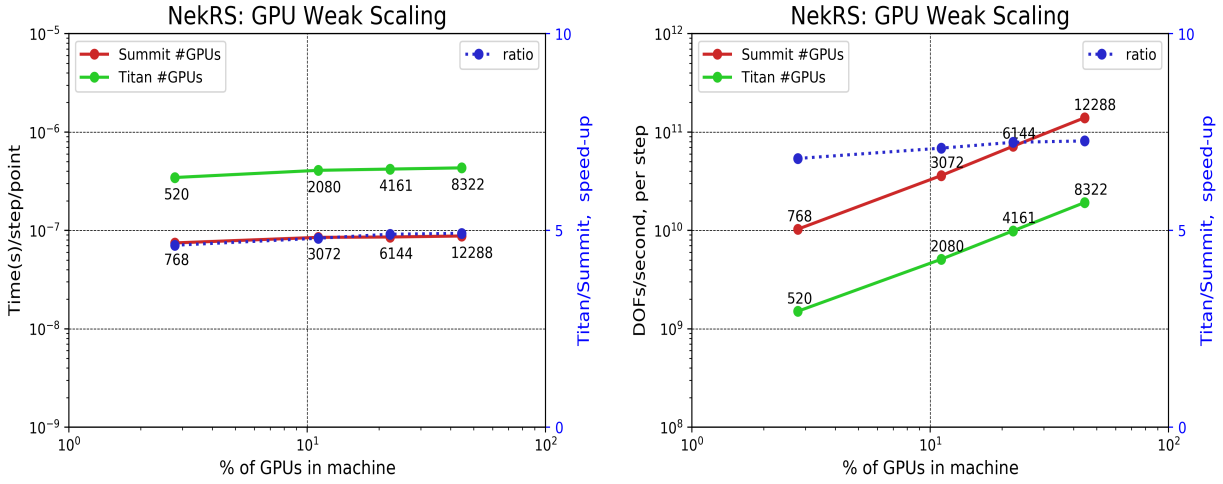


Figure 10: Nek5000+libParanumal (NekRS) performance baseline on Summit.

Runge-Kutta scheme and measured maximum size per GPU, $n = E(N+1)^3$ with $N = 11$, keeping the ratio 2.777 (5000/1800). We had $n = 5000 \times 12^3 = 8.6$ million grid points per GPU on Summit and $n = 1800 \times 12^3 = 3.1$ million grid points per GPU on Titan with the total degree of freedoms $DOFs = 6 \times n \times$ (#GPUs)) for the 6 components from electric and magnetic fields. Titan's #GPUs is chosen 76% of Summit,

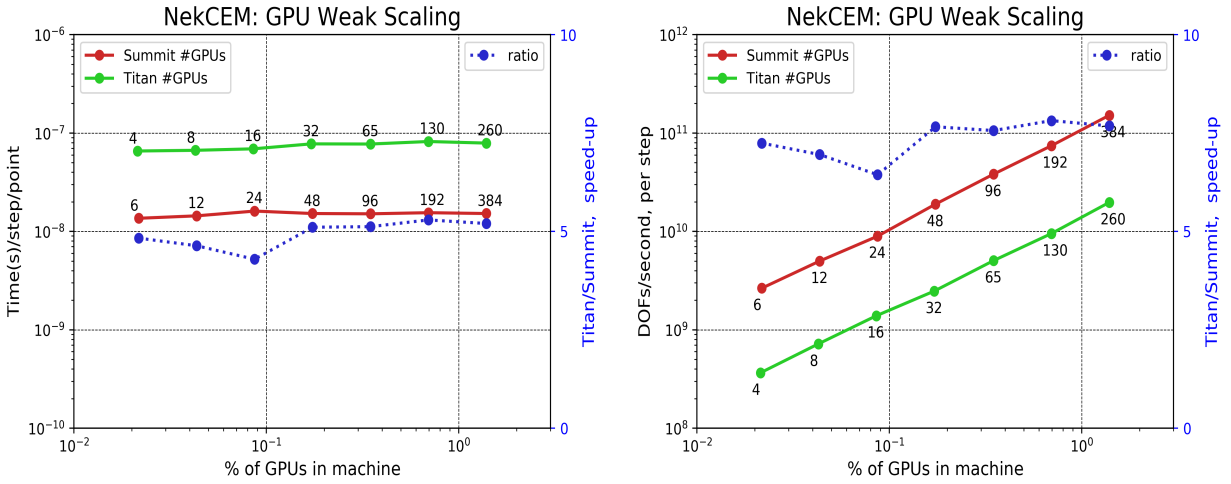


Figure 11: NekCEM performance baseline on Summit.

reflecting full machine GPU ratios. We achieve 5.2× speedup in simulation time and 7.6× speedup in DOFs/sec per step as shown in Figure 11.

Table 2: NekCEM run for 1000 timesteps with 5-stage 4th-order Runge-Kutta time stepping. The ratio of data size 2.777 (5000/1800). Performance is shown in Figure 11.

system	E per GPU	N	$n = E(N+1)^3$	$DOFs = 6 \times n$ per GPU
Summit	5000	11	8.6 M	51.6 M
Titan	1800	11	3.1 M	18.1 M

4.3 ExaSMR: Speedup of NekRS on Summit vs. Nek5000 on Titan

In responding to the need of ExaSMR team, we measured performance baseline the speedup from the newly developed NekRS vs the previously developed OpenACC-based Nek5000. Table 3 shows the experimental setup for Nek5000 OpenACC version.

Figure 12 demonstrates the current baseline measurement showing $\sim 11\times$ speedup from newer version of NekRS on Summit, compared to the old Nek5000 OpenACC version on Titan.

However we used less than 10% for the Nek5000 OpenACC version. With the recent AMG support enabled in Nek5000 OpenACC version, CEED team plans to measure with larger elements counts using 50% of Titan and Summit for 17×17 rods geometry, in collaboration with ExaSMR team.

Our testing setups for baseline performance of Nek5000 on Summit and Titan are summarized in Table 3. We performed Nek5000 OpenACC version running 100 timesteps of eddy simulations without fixing iterations and measured maximum size per GPU, $n = EN^3$ with $N = 14$, keeping the ratio 3.222 (1600/500). We used $n = 1600 \times 14^3 = 3.6$ million grid points per GPU on Summit and $n = 500 \times 14^3 = 3.1$ million grid points per GPU on Titan. The total degree of freedoms is $DOFs = 4 \times n \times$ (#GPUs)) with 4 fields (3 velocity components and 1 pressure). We chose Titan's #GPUs as 76% of Summit.

4.4 ExaSMR: Speedup of Steady Thermal Solver vs. Transient Calculation

In collaboration with ExaSMR team, we performed the spectral element simulations with Nek5000 for all LES, RANS, and pseudo-RANS simulations. The recently proposed Nek5000 steady-state solver has been used for solving the temperature field in the pseudo-RANS approach and has proved significantly faster than transient schemes. Prediction of thermal quantities is compared with classical linear and nonlinear RANS

Table 3: Nek5000 OpenACC version run for 100 timesteps of eddy simulations, without fixing iterations. The ratio of data size 3.222 (1600/500). Performance is shown in Figure 12.

system	E per GPU	N	$n = EN^3$	$DOFs = 4 \times n$ per GPU
Summit	1600	14	8.6 M	34.4 M
Titan	500	14	3.1 M	12.4 M

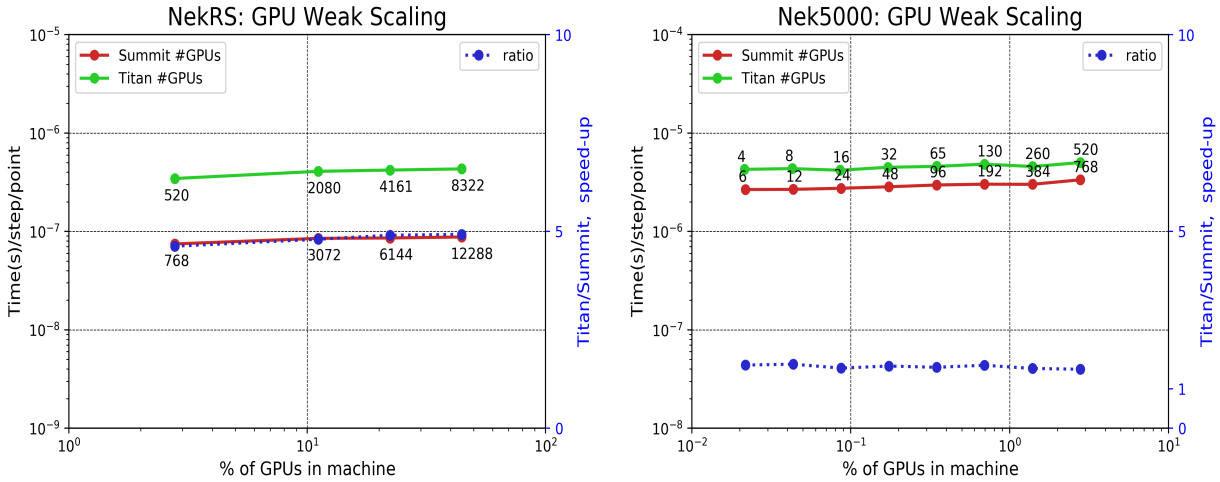


Figure 12: NekRS vs. Nek5000 performance baseline.

models. LES for the full-length rods has also been performed and are used as a reference. Results of the proposed method are demonstrated in Table 4 with significant improvements ($\sim 7\times$) with respect to those obtained with RANS.

Table 4: Time and # iterations required to reach L_2 errors at the level of 10^{-7}, using 8 KNL nodes (512 MPI ranks) on Argonne's LCRC/Bebop.

	# iter			CPU time [min]		
Case	$tol = 10^{-2}$	$tol = 10^{-4}$	$tol = 10^{-6}$	$tol = 10^{-2}$	$tol = 10^{-4}$	$tol = 10^{-6}$
BDF1	75275	34456	23026	154	97	114
BDF2	63854	31854	22371	130	90	106
BDF3	57163	30063	22095	117	85	103
SS	60			12		

5. OTHER PROJECT ACTIVITIES

5.1 Laghos CTS-2 benchmark

The CEED team at LLNL developed a version of the Laghos miniapp to be used in the second edition of the Commodity Technology Systems procurement process. These systems leverage industry advances and open source software standards to build, field, and integrate Linux clusters of various sizes into production service. The CTS-2 version of Laghos features RAJA backend, improved robustness and figure of merit computation, and is available at `https://github.com/CEED/Laghos/releases/tag/cts2`.

5.2 Outreach

CEED researchers were involved in a number of outreach activities, including participation in the SIAM CSE19 conference, where CEED organized two minisymposiums (16 talks total) on Exascale Software for High-Order Methods and Exascale Applications with High-Order Methods and CEED researchers participated in various additional minisymposiums related to Batched BLAS, meshing, DG methods, matrix-free solvers, and more. CEED organized a successful breakout session on high-order methods and application at the ECP third annual meeting in Houston and we begun the planning for CEED's 3rd annual meeting to be held at Virginia Tech in August 2019.

6. CONCLUSION

The deliverable for this milestone was the release of CEED 2.0 – the CEED software distribution consisting of integrated libraries and software packages enabling efficient high-order discretizations on unstructured grids provided through the CEED website, the CEED GitHub organization, and Spack.

The artifacts delivered include a consistent build system based on 12 Spack packages + a the CEED meta-package, documentation and verification of the build process, as well as improvements in the integration between different CEED components. See the CEED website, `http://ceed.exascaleproject.org/ceed-2.0/` and the CEED GitHub organization, `http://github.com/ceed` for more details. As part of CEED 2.0, we also released the next version of libCEED (v0.4), which contains four new CPU backends, two new GPU backends, CPU backend optimizations, initial support for operator composition, performance benchmarking, and a Navier-Stokes demo.

In addition to details and results from these efforts, in this document we report on other project-wide activities performed in Q2 of FY19, including: a version of the Laghos miniapp for CTS-2 procurement, extensive GPU performance baseline benchmarking for the Urban and ExaSMR applications, papers, minisimposium presentations, 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-771107.