LibCEED 0.8: Concepts and mini-apps

Contact Information:
https://ceed.exascaleproject.org
https://github.com/CEED/libCEED

email: jed@jedbrown.org

Valeria Barra[1,2], Natalie Beams[3], Jed Brown[1], Yohann Dudouit[4], Leila Ghaffari[1], Arash Mehraban[1], Will Pazner[4], Rezgar Shakeri[1], and Jeremy Thompson[1]

[1]Department of Computer Science, University of Colorado Boulder
[2]Department of Environmental Science and Engineering, California Institute of Technology
[3]Innovative Computing Laboratory, University of Tennessee
[4]Lawrence Livermore National Laboratory

2021 ECP Annual Meeting

Abstract

LibCEED is a new open-source mathematical software library that provides a purely algebraic interface for optimized matrix-free representation and preconditioning of linear and nonlinear operators tuned for a variety of computational device types, including CPUs and GPUs. LibCEED is independent of a specific application and is minimally intrusive, which makes it easy to port to new applications.

New Features

- Julia and Rust interfaces added.
- Static libraries can be built with `make STATIC=1` and the pkg-config file is installed accordingly.
- Added support for full assembly of libCEED operators.
- New HIP MAGMA backends for hipMAGMA library users: `/gpu/hip/magma` and `/gpu/hip/magma/det`.
- New HIP backends for improved tensor basis performance: `/gpu/hip/shared` and `/gpu/hip/gen`.
- Expanded Fluid and Solid mechanics examples.

Operator Decomposition

Finite element operators are typically defined through weak formulations of PDEs that involve integration over a computational mesh. The required integrals are computed by splitting them as a sum over the mesh elements, mapping each element to a simple reference element, and applying a quadrature rule in the reference space.

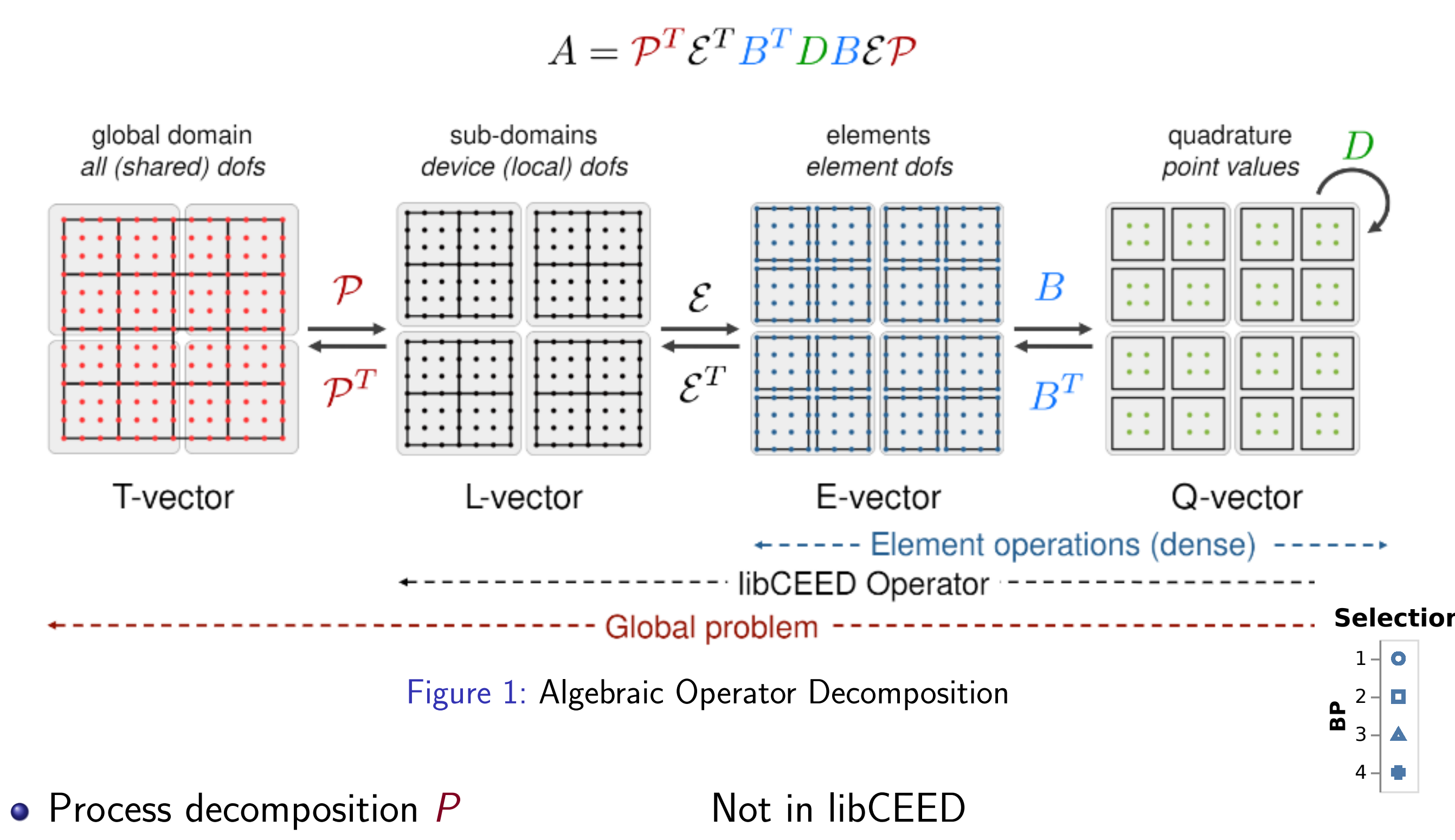

Figure 1: Algebraic Operator Decomposition

• Process decomposition $\mathcal{P}$	Not in libCEED
• Element restriction $\mathcal{E}$	CeedElemRestriction
• Basis (Dofs-to-Qpts) evaluator B	CeedBasis
• Operator at quadrature points D	CeedQFunction
• $A_L = \mathcal{E}^T B^T D B \mathcal{E}$	CeedOperator

LibCEED also supports composition of different operators for multiphysics probler mixed-element meshes which is provided by **Composite Operator**.

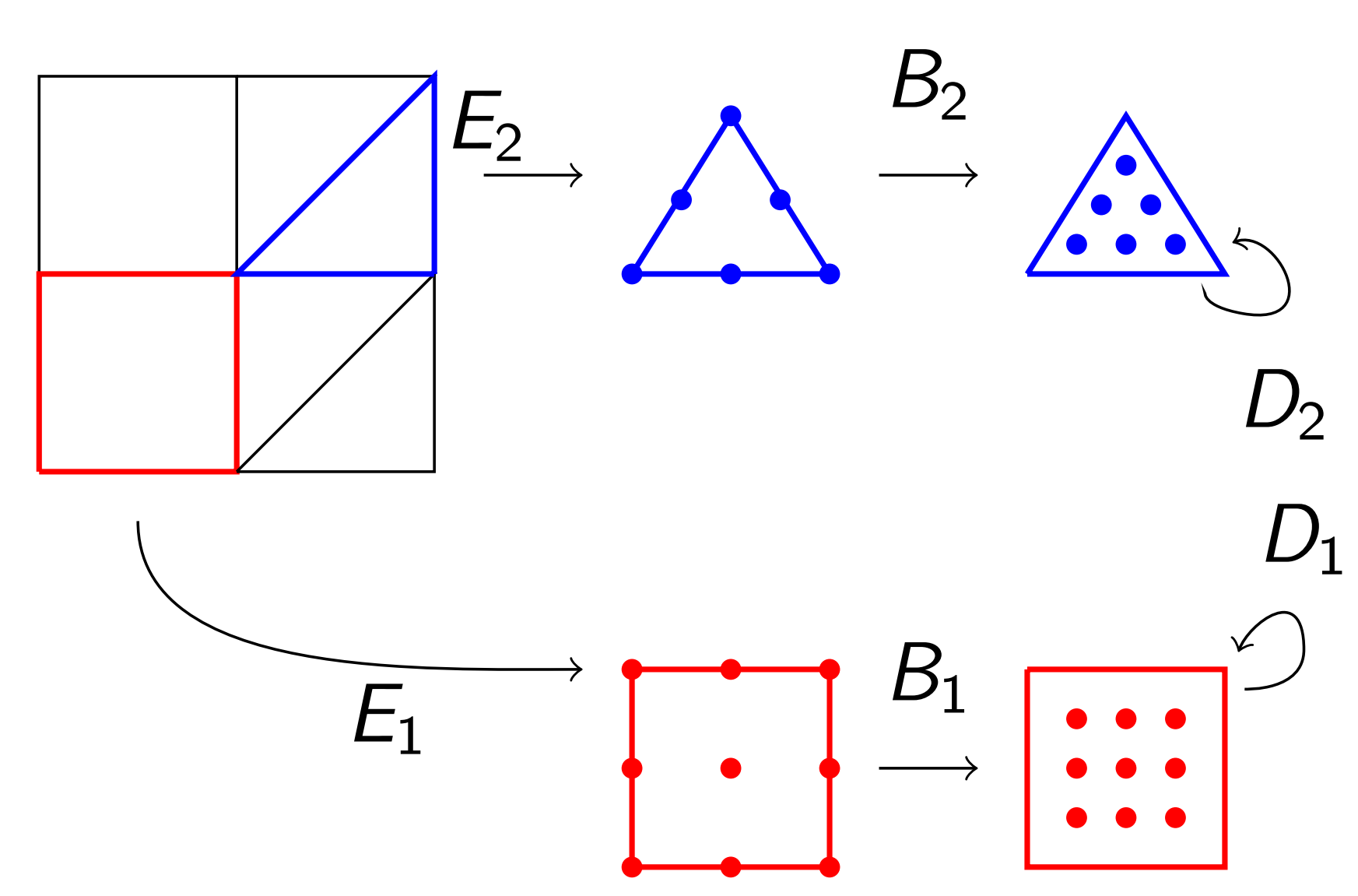

Figure 2: A schematic of element restriction and basis applicator operators for elements with different topology. The same CeedQFunction can be used with different topology and degree elements.

Backends

The libCEED API takes an algebraic approach, where the user describes in the frontend the operators $\mathcal{E}$, B, and D and the library provides backend implementations and coordinates their action to the original operator independently on each device/MPI task. This purely algebraic description includes all the finite element information, so backends can operate on the linear algebra level without explicit finite element code. The separation of the frontend and backends enables applications to easily change backends.

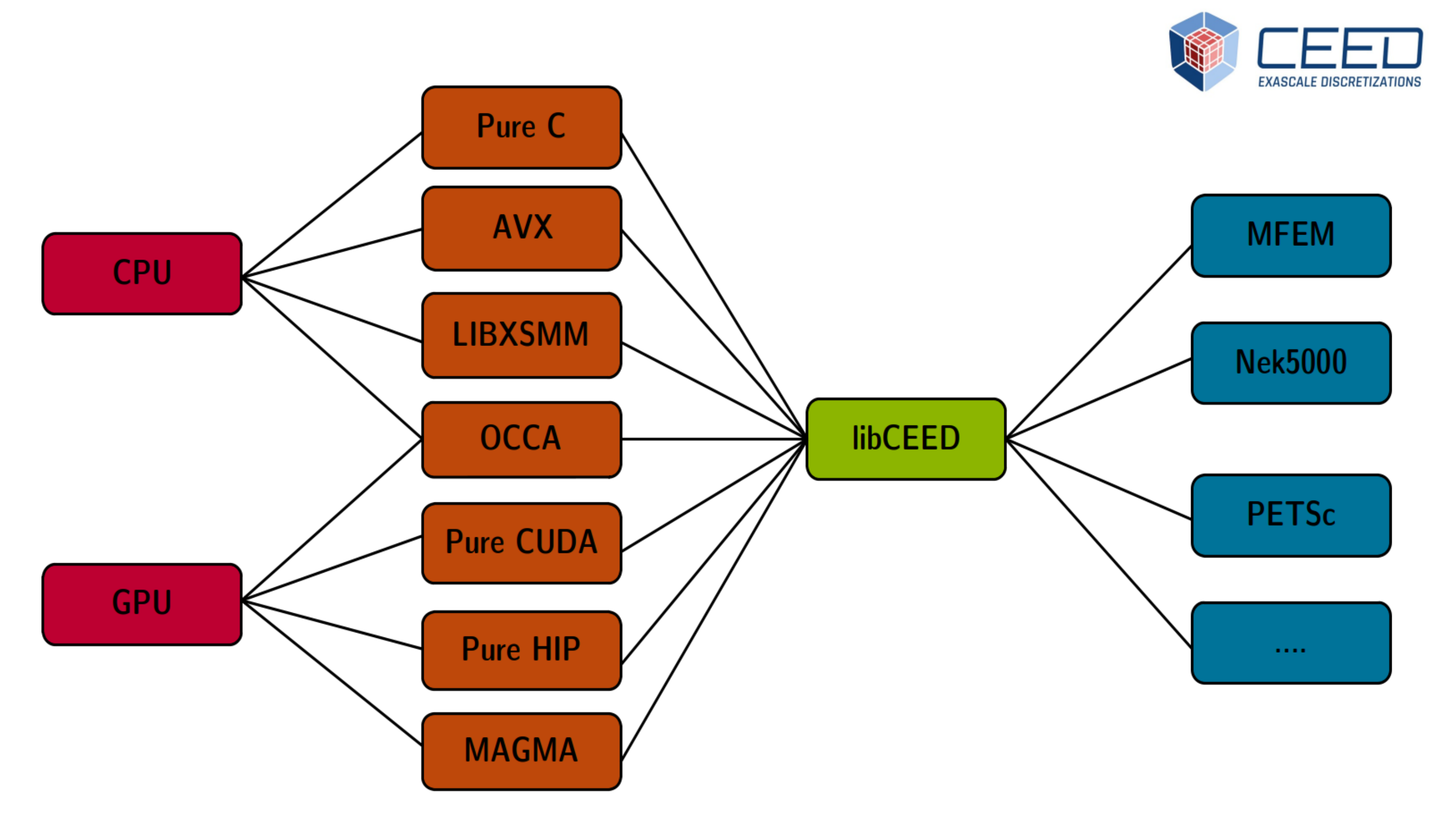

Figure 3: libCEED Backends

Performance

The CEED project uses Benchmark Problems (BP) to test and compare the performance of high order finite element codes. We analyze the performance on BP3, **Poisson problem with homogeneous Dirichlet boundary conditions**. We measure performance over 20 iterations of unpreconditioned Conjugate Gradient (CG) on hexahedral 3D elements with 1 more quadrature point than the number of nodes for the shape function in 1D.
The **LIBXSMM** backends offer the best performance on the CPU and the **CUDA** backend with **code generation** offers the best performance on the GPU. The plots below show work, measured by DoFs multiplied by CG iterations divided by compute nodes multiplied by seconds, plotted against time on the left plot where on the right plot, it is plotted against problem size, measured by points per compute node.

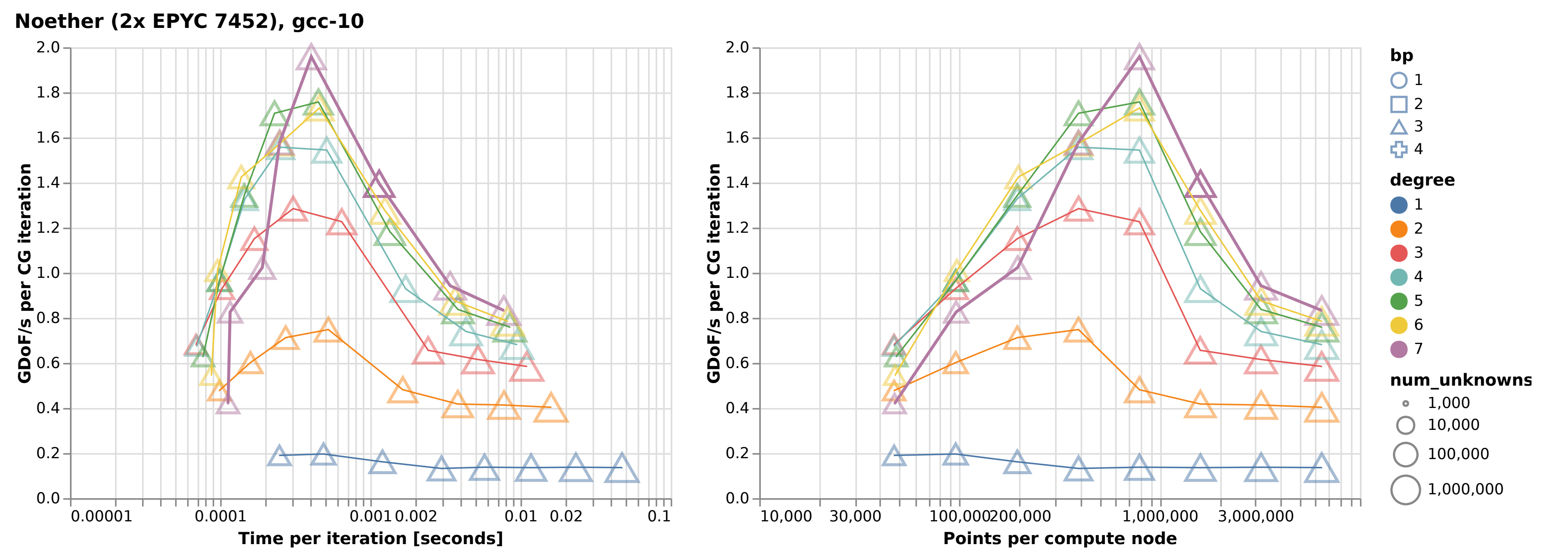

Figure 4: 2x AMD EPYC 7452 (32-core) with gcc-10 compiler. LIBXSMM blocked backend ($q = P + 1$, $P = p + 1$) with respect to time (left) and problem size (right)

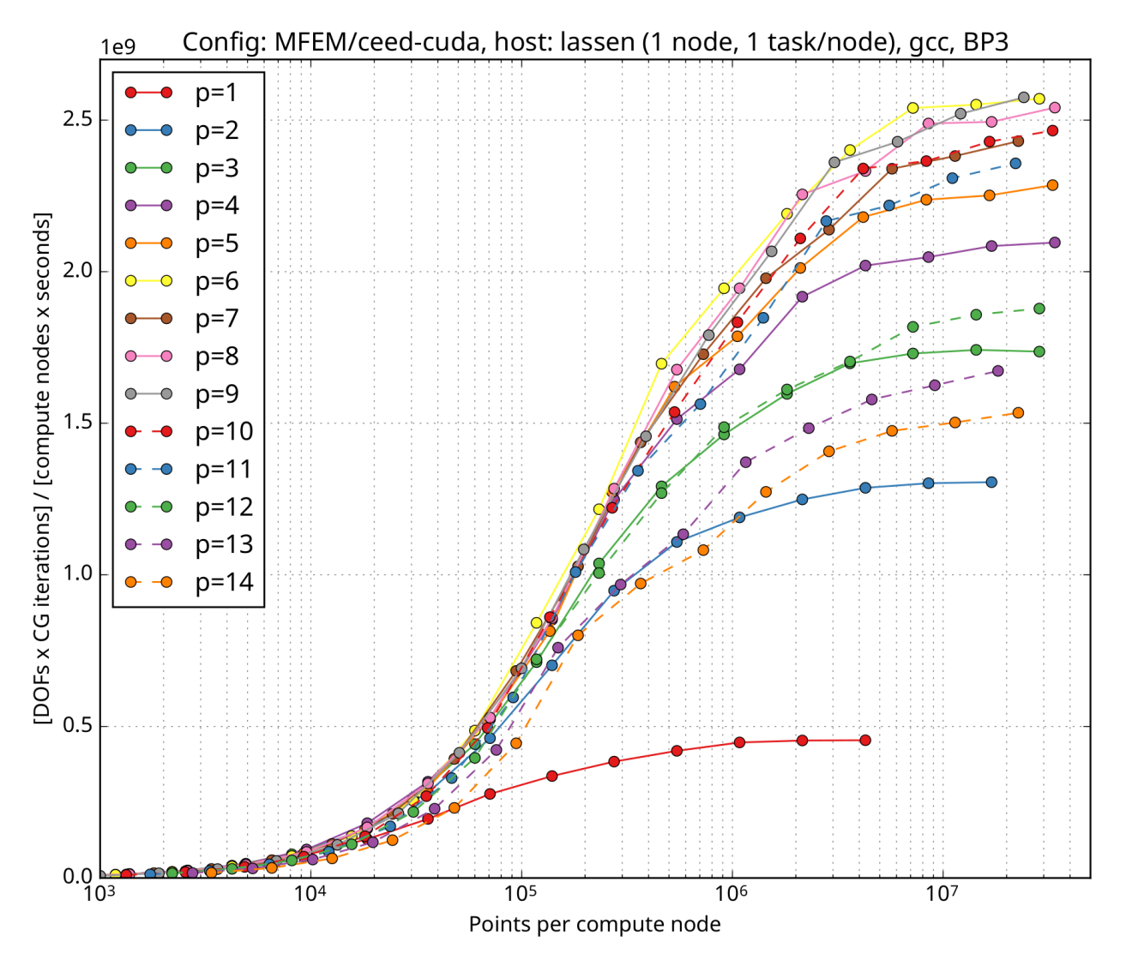

Figure 5: CUDA-gen backend performance for BP3 on NVIDIA V100.

NOTE!

The peak performance obtained on the CPU is competitive with the results obtained on the GPU.

Application - Solid Mechanics

Finite strain Neo-Hookean hyperelasticity are implemented in current configuration by pushing forward the residual and Jacobian operator in initial configuration as written in equations (1) and (2) respectively.

$$\underbrace{\nabla_X \boldsymbol{v} : \boldsymbol{F}\boldsymbol{S}}_{\text{Initial Residual}} \xrightarrow{\text{push forward}} \underbrace{\nabla_x \boldsymbol{v} : \boldsymbol{\tau}}_{\text{Current Residual}} \tag{1}$$

$$\underbrace{\nabla_X \boldsymbol{v} : \left(\mathrm{d}\boldsymbol{F}\boldsymbol{S} + \boldsymbol{F}\mathrm{d}\boldsymbol{S}\right)}_{\text{Initial Jacobian}} \xrightarrow{\text{push forward}} \underbrace{\nabla_x \boldsymbol{v} : \left(\mathrm{d}\boldsymbol{\tau} - \boldsymbol{\tau}(\nabla_x \mathrm{d}\boldsymbol{u})^T\right)}_{\text{Current Jacobian}} \tag{2}$$

where $\boldsymbol{F} = \boldsymbol{I}_3 + \nabla_X \boldsymbol{u}$ is the gradient deformation, $\boldsymbol{S}$ is second Piola-Kirchoff tensor, $\boldsymbol{\tau} = \boldsymbol{F}\boldsymbol{S}\boldsymbol{F}^T$ is Kirchoff stress tensor and $\mathrm{d}(\cdot)$ denotes the directional derivatives.
In the following table the SNES solver time for the finite element basis of degree $p = 3$ is shown for four storage variants. Note that static storage are the data for the mapping between reference element coordinate $\hat{X}$ and physical coordinate system and computed storage are the variables that are stored during the residual evaluation. In initial configuration, the solver time can be reduced at the cost of more storage (26 scalars per quadrature point). However, with 17 scalars storage in the current configuration we can achieve the minimum cost of computation.

Option `-problem`	Static storage	Computed storage	# scalars	Solve time
`FSInitial-NH1`	$\nabla_X \hat{X}$, $\det \nabla_{\hat{X}} X$	$\nabla_X \boldsymbol{u}$	19	7.03 sec
`FSInitial-NH2`	$\nabla_X \hat{X}$, $\det \nabla_{\hat{X}} X$	$\nabla_X \boldsymbol{u}$, $\boldsymbol{C}^{-1}$, $\lambda \log J$	26	7.02 sec
`FSCurrent-NH1`	$\nabla_X \hat{X}$, $\det \nabla_{\hat{X}} X$	$\nabla_X \boldsymbol{u}$	19	6.69 sec
`FSCurrent-NH2`	$\det \nabla_{\hat{X}} X$	$\nabla_x \hat{X}$, $\boldsymbol{\tau}$, $\mu - \lambda \log J$	17	5.79 sec

Application - Fluid Mechanics

Isentropic traveling vortex test case is an analytical solution to the Euler equations with perturbation in density, velocity, and temperature while entropy remains constant ($S = 1$). We have applied in/outflow BCs using the **Composite Operator** in libCEED where we applied the exact solution on the inflow BCs and the evaluated solution on the outflow boundaries. This example is useful for testing boundary conditions, discretization stability, and order of accuracy.

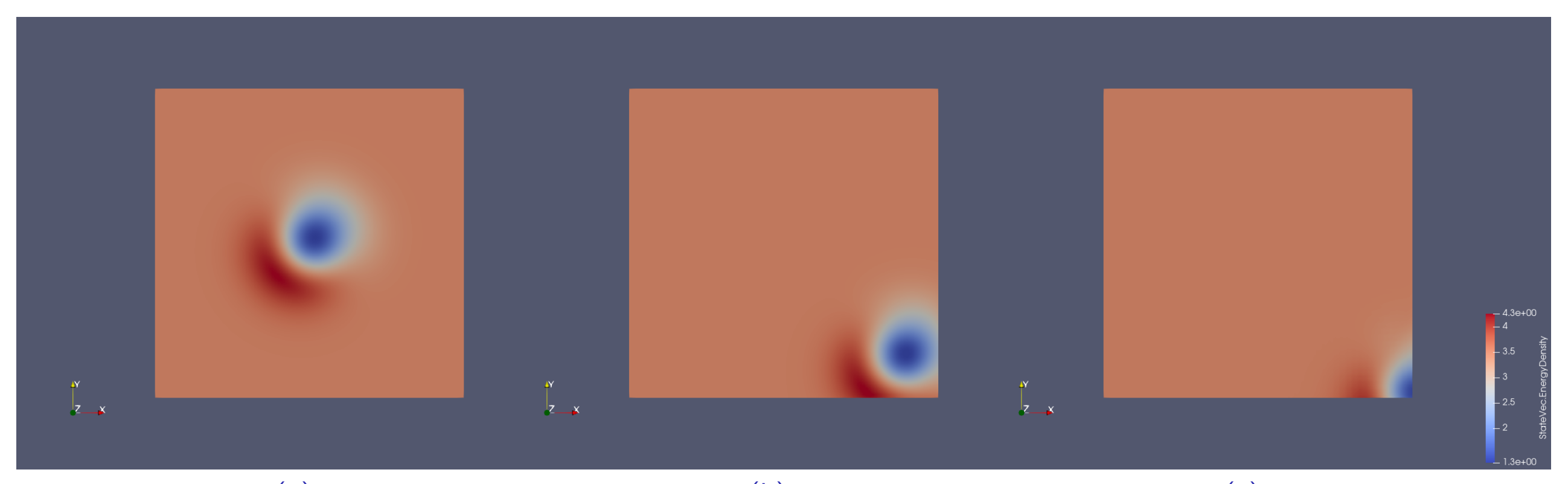
(a) (b) (c)

Figure 6: Isentropic vortex on its way leaving the domain.

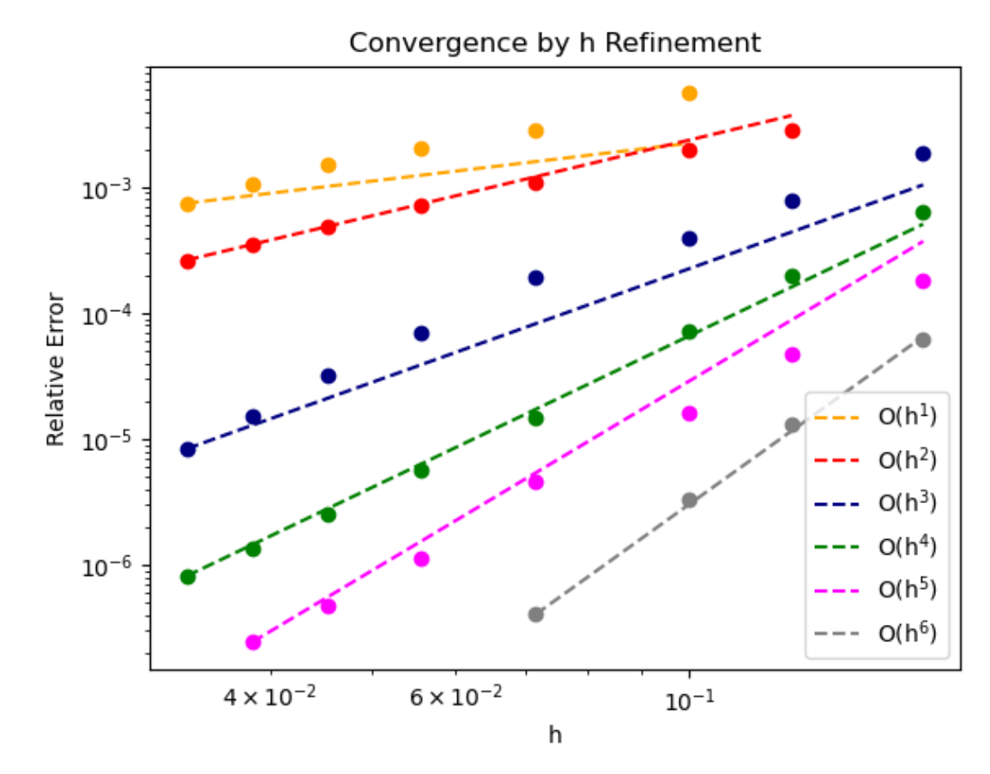

Figure 7: Convergence plot

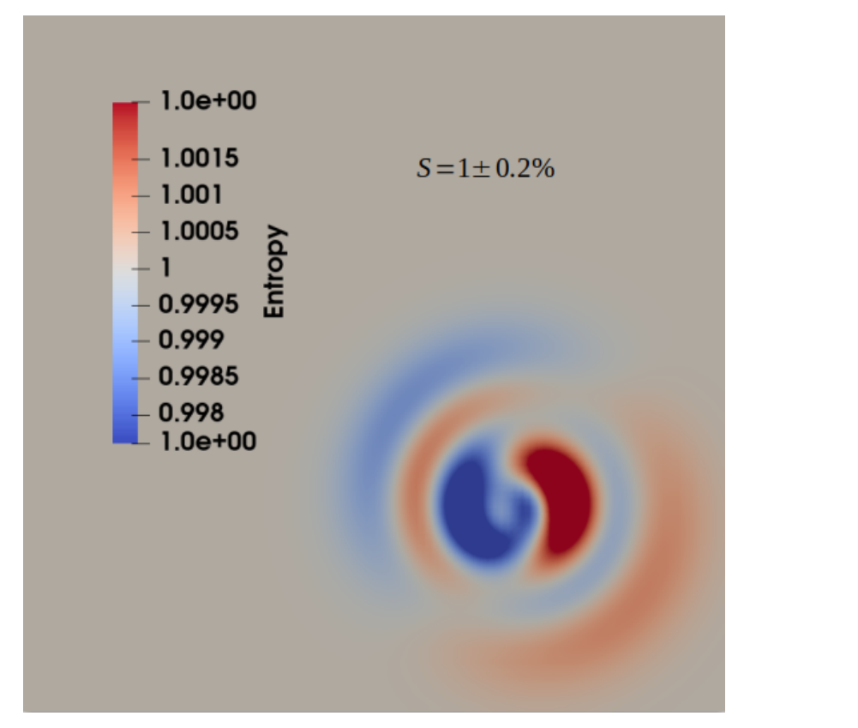

Figure 8: Constant entropy

Outlook

- Backend optimization
- Algorithmic differentiation of Q-functions
- Higher level interfaces (MFEM, PETSc)
- Expanding the application examples

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.