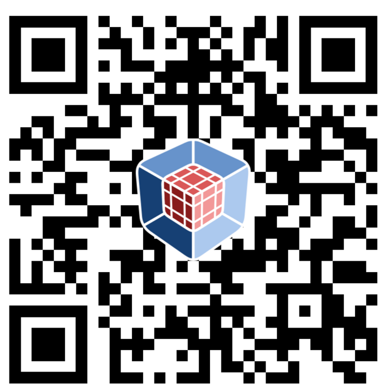

Contact Information:
https://ceed.exascaleproject.org
https://github.com/CEED/libCEED
email: `valeria.barra@colorado.edu`
email: **jeremy.thompson@colorado.edu**

libCEED - Lightweight High-Order Finite Elements Library with Performance Portability and Extensibility

Jeremy Thompson[1], Valeria Barra[2], Yohann Dudouit[3], Oana Marin[4] & Jed Brown[2]

1: Department of Applied Mathematics & 2: Computer Science, University of Colorado Boulder
3: Lawrence Livermore National Laboratory, 4: Argonne National Laboratory

CEED
EXASCALE DISCRETIZATIONS

Abstract

High-order numerical methods are widely used in PDE solvers, but software packages that have provided high-performance implementations have often been special-purpose and intrusive. libCEED is a new library that offers a **purely algebraic interface** for **matrix-free operator** representation and supports run-time selection of implementations tuned for a variety of computational device types, including **CPUs and GPUs**. We introduce the libCEED API and demonstrate how it can be used in standalone code or integrated with other packages (e.g., PETSc, MFEM, Nek5000) to solve examples of problems that often arise in the scientific computing community, ranging from fast solvers via geometric multigrid methods to Computational Fluid Dynamics (CFD) applications.

Operator Decomposition

Finite element operators are typically defined through weak formulations of PDEs involving 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.
This is illustrated below for a symmetric linear operator on third order (Q3) scalar continuous (H1) elements, where T-vector, L-vector, E-vector and Q-vector represent the (true) degrees of freedom on the global mesh, the split local degrees of freedom on the subdomains, the split degrees of freedom on the mesh elements, and the values at quadrature points, respectively.

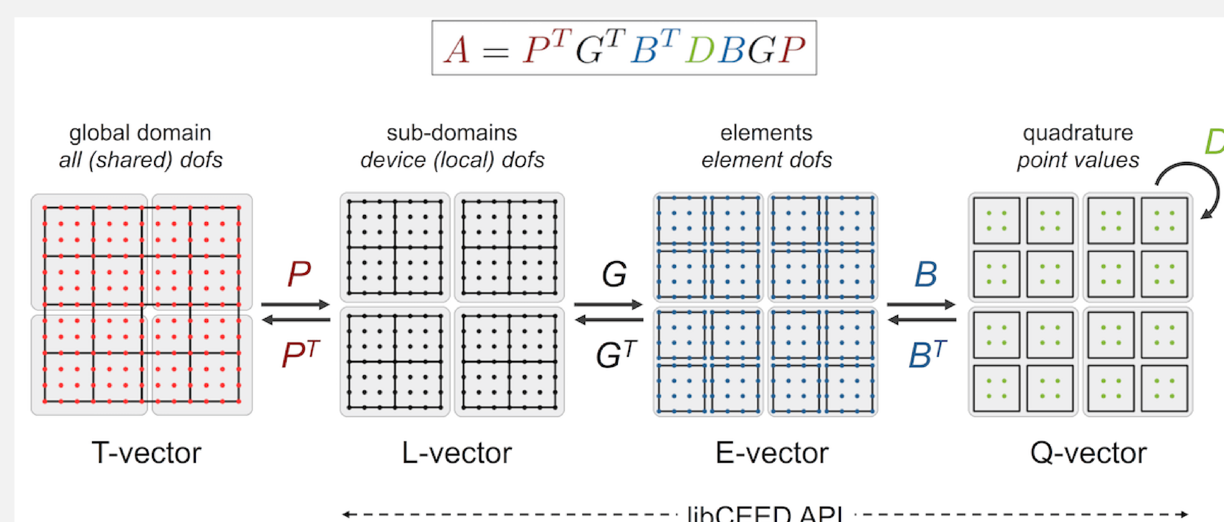

Figure 1: Operator Algebraic Decomposition

- Process decomposition P — Not in libCEED
- Element restriction G — CeedElemRestriction
- Basis (Nodes-to-Qpts) evaluator B — CeedBasis
- Operator at quadrature points D — CeedQFunction
- $A_L = G^T B^T D B G$ — CeedOperator

This decomposition exposes **optimizations for modern architectures, high-order elements, and tensor product elements** not available with sparse matrices.

Extensible Backends

The libCEED API takes an algebraic approach. The user describes the objects G, B, and D; and the library provides backend implementations to apply the local action of the PDE operator A_L. This **purely algebraic description** includes all 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**.

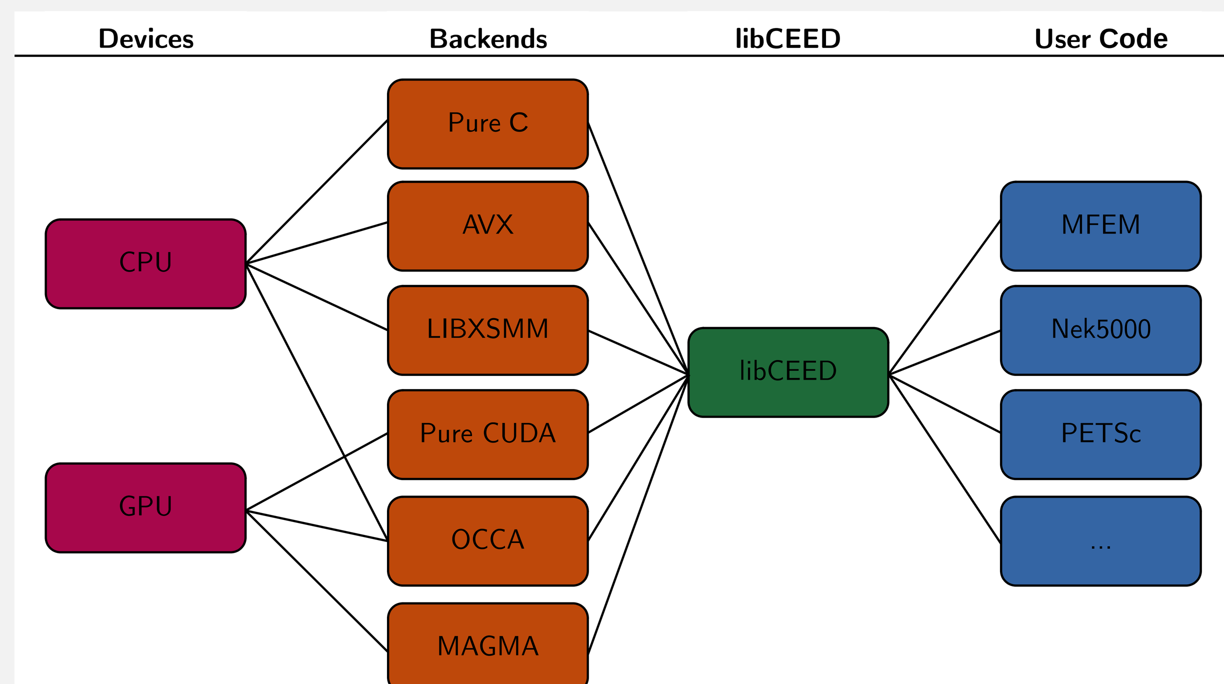

Figure 2: libCEED Backends

libCEED supports several backends and users can develop additional backends. **The LIBXSMM backends offer the best performance on the CPU and the CUDA backend with code generation offers the best performance on the GPU.**

Performance Benchmarks

The CEED project uses Benchmark Problems (BP) to test and compare the performance of high order finite element codes. We analyze the performance of libCEED backends 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 **plots below show work**, measured by DoFs multiplied by CG iterations divided by compute nodes multiplied by seconds, **plotted against problem size**, measured by points per compute node. A variety of polynomial orders of the 1D shape functions, p, are shown.

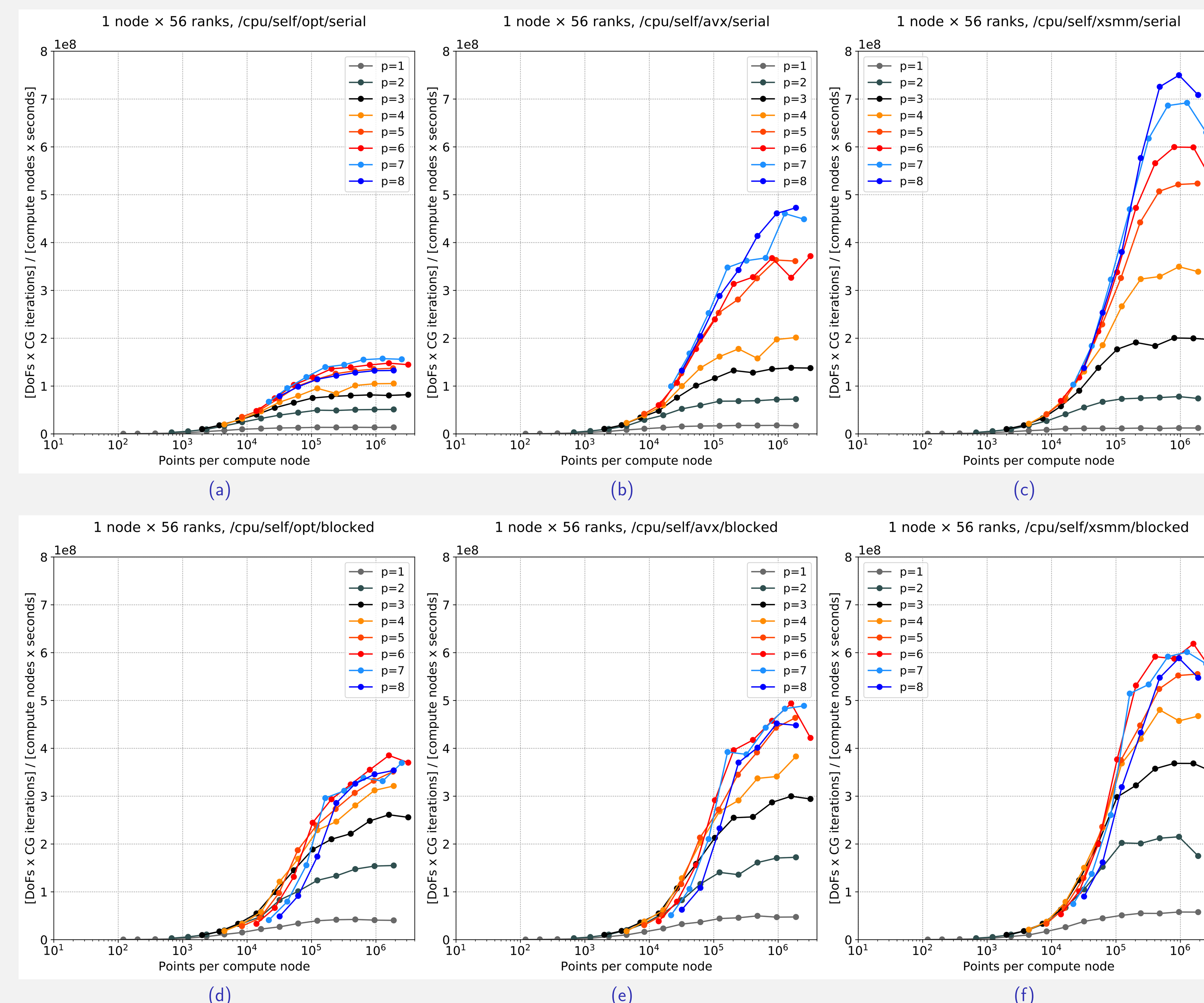

Figure 3: BP3: 2x Intel Xeon Platinum 8180M CPU 2.50GHz (Skylake): In (a)-(c), internal vectorization implementation. In (d)-(f), external vectorization implementation.

These benchmark codes are included in libCEED's PETSc example suite. The top row shows the performance of CPU backends utilizing an internal vectorization strategy for element basis operation (B) and QFunction (D) evaluation, processing on element at a time. The second row shows CPU backends utilizing an external vectorization strategy, processing batches of 8 elements at a time with data interlaced to provide vectorization friendly lengths for the tensor contractions and QFunction evaluations. **For lower order elements, the external vectorization is more efficient, while at higher orders the elements are sufficiently large that internal vectorization is more efficient**, due to cache size limitations.

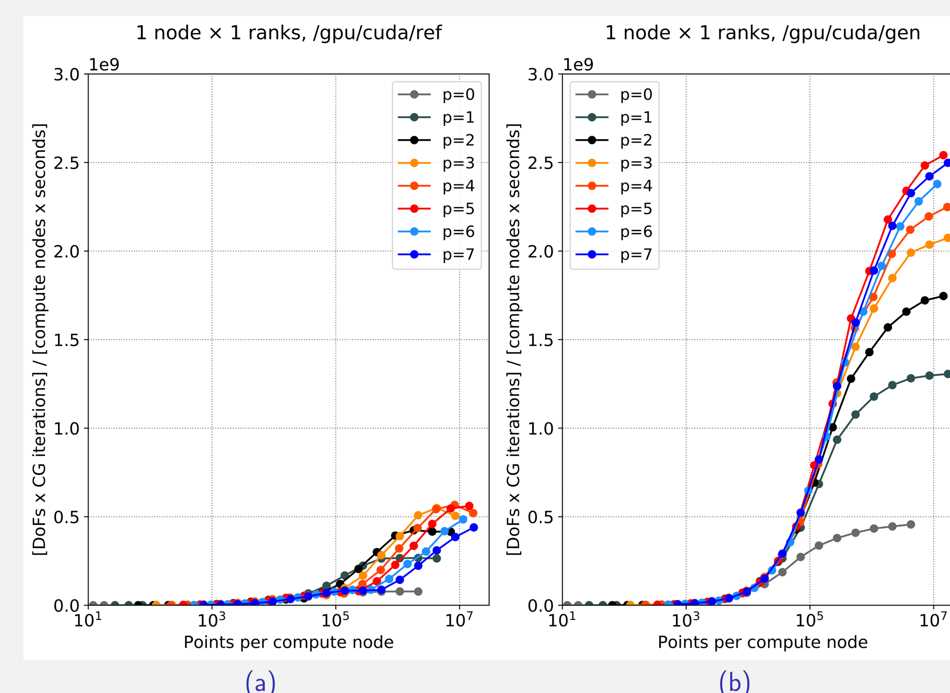

Figure 4: BP3: 1x NVIDIA V100 GPU (Volta): In (a) reference CUDA implementation. In (b) CUDA code generation implementation.

These benchmarks use MFEM. The CUDA code generation backend, /gpu/cuda/gen, uses the user provided QFunction (D) code and information in the basis (B) and element restriction (G) objects to provide a single local operator kernel (A_L). This significantly improves performance over launching separate kernels to handle the action of each API object, seen in /gpu/cuda/ref. **The code generation in /gpu/cuda/gen can provide +/- 10% performance seen in hand written CUDA code, such as the code provided in libParanumal.** Collaboration with the libParanumal team provided several performance enhancements to the libCEED CUDA backends.

Application - Geometric Multigrid

With high order finite elements, **preconditioning is essential to control the condition number and total iteration count** for iterative solvers such as CG. The libCEED operator decomposition offers several opportunities for preconditioning. P-multigrid offers mesh independent convergence for unstructured meshes. Restriction, Prolongation, and Smoothing operators can all be implemented in libCEED. We investigate performance of p-multigrid for BP3 on an unstructured mesh with hexahedral 3D elements.

- Prolongation/Interpolation Operator
  $A_L = G_c^T I I B_{ftoc} G_f$
- Laplacian Operator
  $A_L = G^T \hat{B}_{grad}^T D \hat{B}_{grad} G$

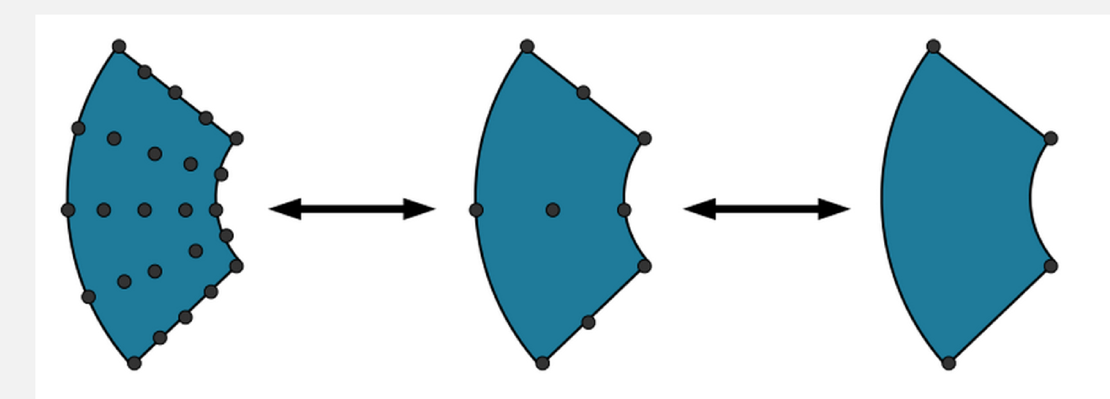

Figure 5: P-Multigrid on Unstructured Mesh

We consider BP3 on 3D hexahedral elements with 7th order polynomial basis functions on an unstructured mesh on a cubic domain with 20^3 elements and 2.69 million DoFs.

Performance	Unpreconditioned	P-Multigrid
$\|\|\cdot\|\|_\infty$ error	$5.08\text{x}10^{-12}$	$3.75\text{x}10^{-12}$
CG Iterations	80	6
CG Solve Time	8.5 sec	8.5 sec

The p-multigrid example is still in development and requires performance tuning to reduce the time per CG iteration; however, initial results demonstrate the flexibility of the libCEED API in offering preconditioning strategies for high-order operators on unstructured meshes.

Application - Navier-Stokes

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 discretization and explicit high-order time-stepping. We solve the density current problem: **a cold air bubble** drops by convection in a neutrally stratified atmosphere. The mathematical formulation is given below. The compressible Navier-Stokes equations in conservative form are

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

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

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

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 $P = (c_p/c_v - 1)(E - \boldsymbol{U} \cdot \boldsymbol{U}/(2\rho) - \rho g z)$ is the pressure, 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).

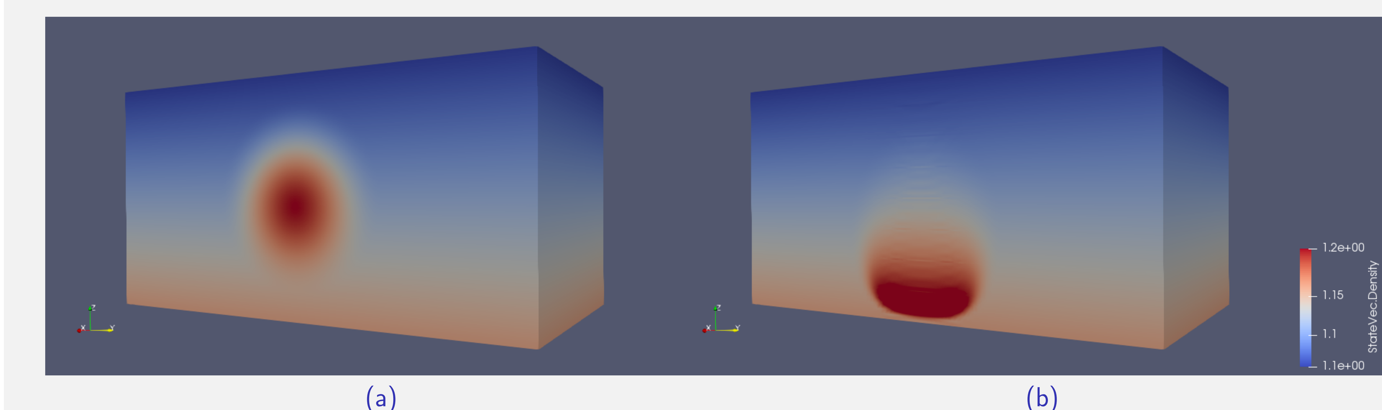

Figure 6: Solution of the compressible Navier-Stokes equations. In (a) the initial condition; In (b) the evolution at time $t = 50$ s.

Development Outlook

Status: Ready for collaborators and friendly users.

- Further performance tuning for CPU and GPU
- Improved non-conforming and mixed-mesh support
- Preconditioning based on libCEED decomposition
- Algorithmic differentiation of Q-functions
- HIP, OpenCL, and OpenMP Backends with OCCA

Grant: Exascale Computing Project (17-SC-20-SC)