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

email: `valeria.barra@colorado.edu`

# Performance Portability with Matrix-Free Finite Element Operators

**Valeria Barra** & Jed Brown & Jeremy Thompson
Department of Computer Science, University of Colorado Boulder

*PETSc User Meeting 2019, Georgia Tech, Atlanta, GA, USA*

## Abstract

One of the challenges with high-order finite element and spectral element methods is that a global sparse matrix is no longer a good representation of a high-order linear operator, both with respect to the FLOPs needed for evaluation and the memory transfer needed for a matrix-vector multiply. Thus, high-order methods require a new operator description that still represents a linear or non-linear operator. libCEED is an extensible library that provides a portable algebraic interface and optimized implementations suitable for high-order operators. libCEED's operator description is easy to incorporate in a wide variety of applications, without significant refactoring of the discretization infrastructure. We introduce the libCEED API, show performance results for CEED Benchmark Problems, and introduce a Navier-Stokes demo solver using libCEED and PETSc.

## 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.
This is illustrated below for the simple case of symmetric linear operator on third order (Q3) scalar continuous (H1) elements, where we use the notions T-vector, L-vector, E-vector and Q-vector to represent the sets corresponding to 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.

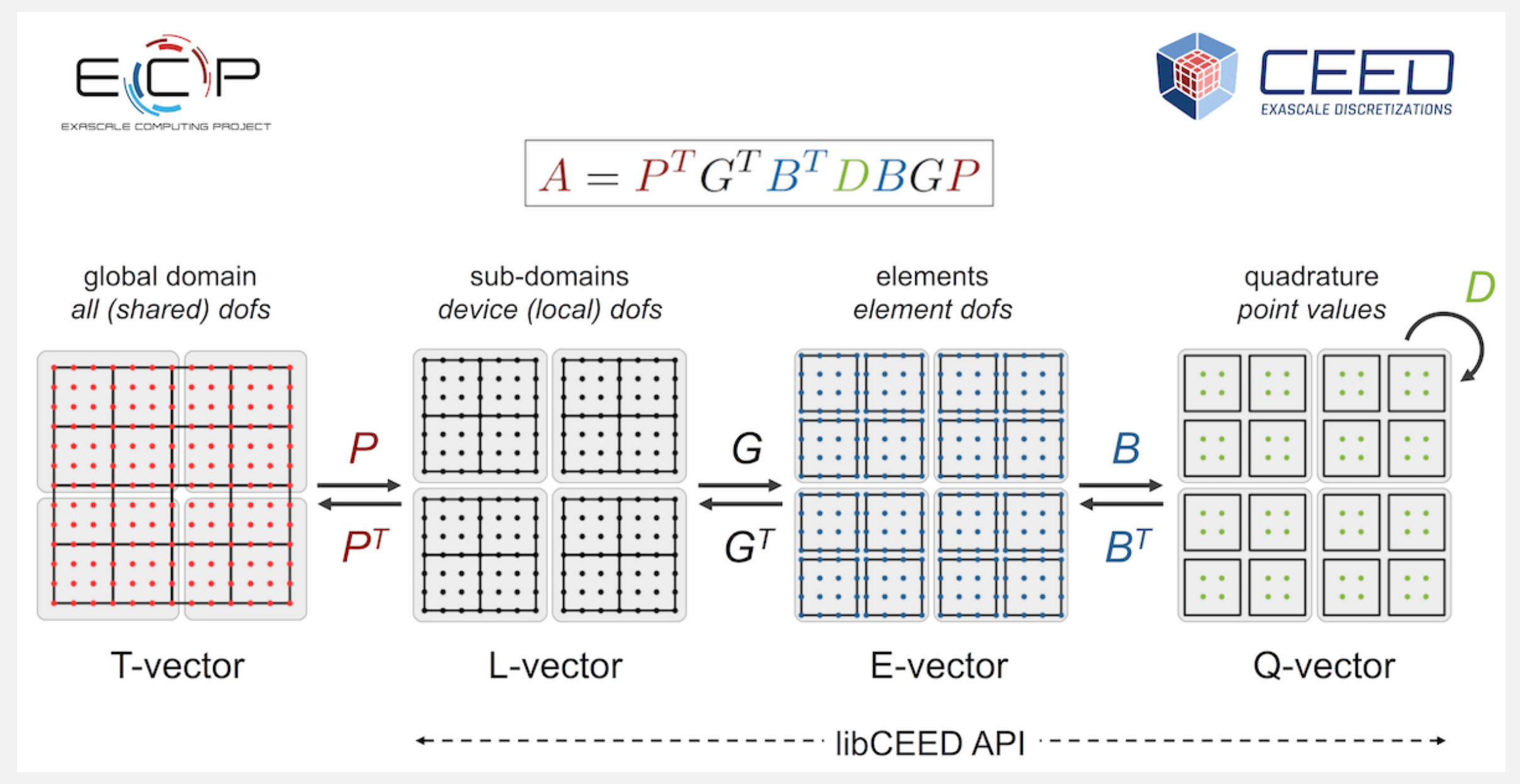

Figure 1: Algebraic Operator Decomposition

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

## Backends

The libCEED API takes an algebraic approach, where the user describes in the frontend the operators $G$, $B$, 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.

| CEED Backend | Description |
|---|---|
| /cpu/self/ref/* | Serial/Blocked reference C implementation |
| /cpu/self/avx/* | Serial/Blocked AVX implementation |
| /cpu/self/xsmm/* | Serial/Blocked LIBXSMM implementation |
| /cpu/self/opt/* | Serial/Blocked optimized C implementation |
| /cpu/occa | Serial OCCA kernels |
| /gpu/occa | CUDA OCCA kernels |
| /omp/occa | OpenMP OCCA kernels |
| /ocl/occa | OpenCL OCCA kernels |
| /gpu/cuda/ref | Reference pure CUDA kernels |
| /gpu/cuda/reg | Pure CUDA kernels, one thread per element |
| /gpu/cuda/shared | Optimized pure CUDA kernels using shared memory |
| /gpu/magma | CUDA MAGMA kernels |

## Performance w.r.t. DOFs

CEED uses Benchmark Problems to test and compare the performance of high order finite element codes.
Benchmark Problem 1 (BP1): $L^2$ projection problem with homogenous Neumann BCs,

$$B\mathbf{u} = \mathbf{f},$$

where $B$ is the mass matrix.
We analyze the performance on BP1 over 20 interations of unpreconditioned Conjugate Gradient on hexahedral 3D elements with 2 more quadrature points than the nodes of the shape function in 1D.

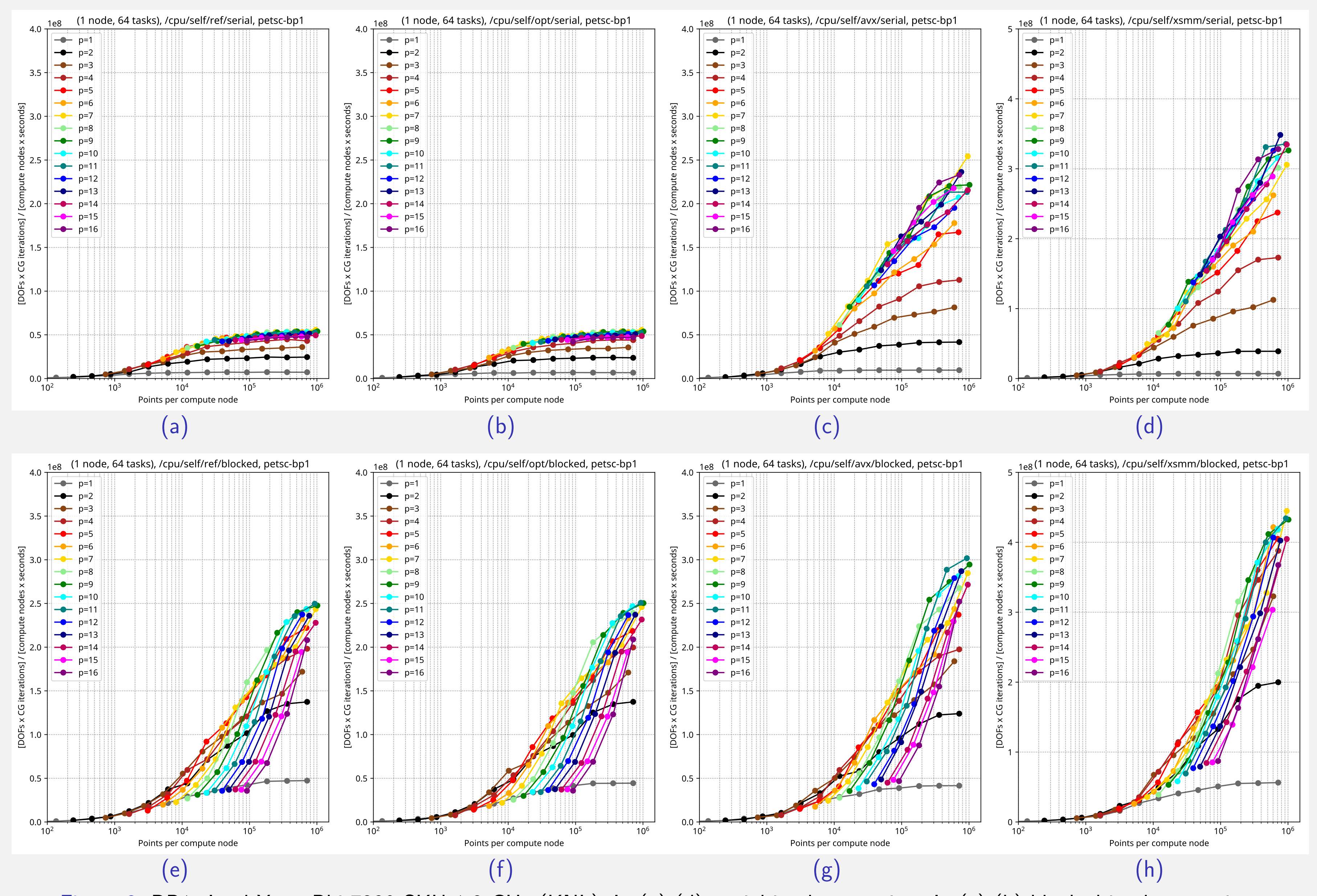

Figure 2: BP1: Intel Xeon Phi 7230 SKU 1.3 GHz (KNL): In (a)-(d), serial implementation. In (e)-(h) blocked implementation.

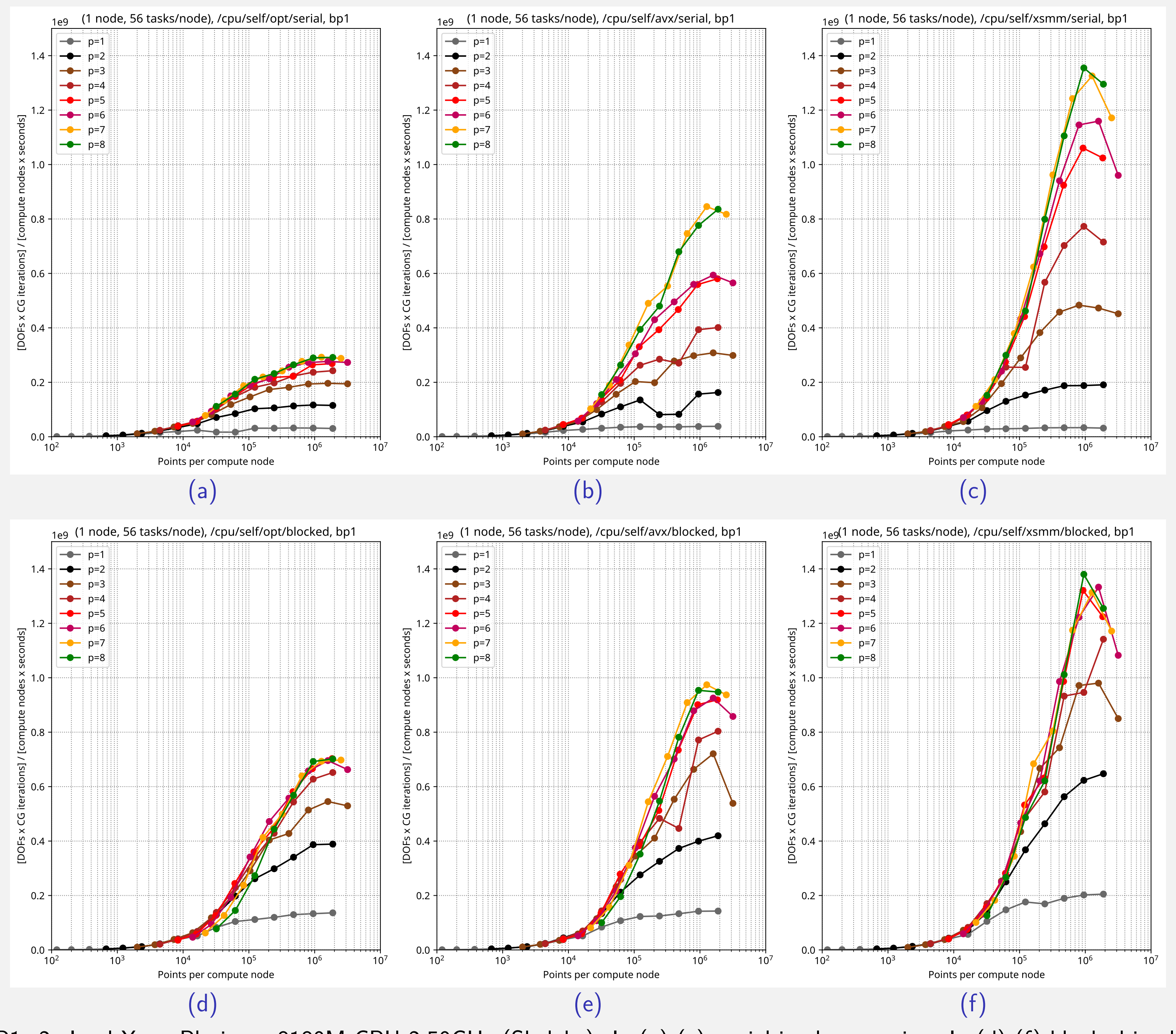

Figure 3: BP1: 2x Intel Xeon Platinum 8180M CPU 2.50GHz (Skylake): In (a)-(c), serial implementation. In (d)-(f) blocked implementation.

## Performance w.r.t. time

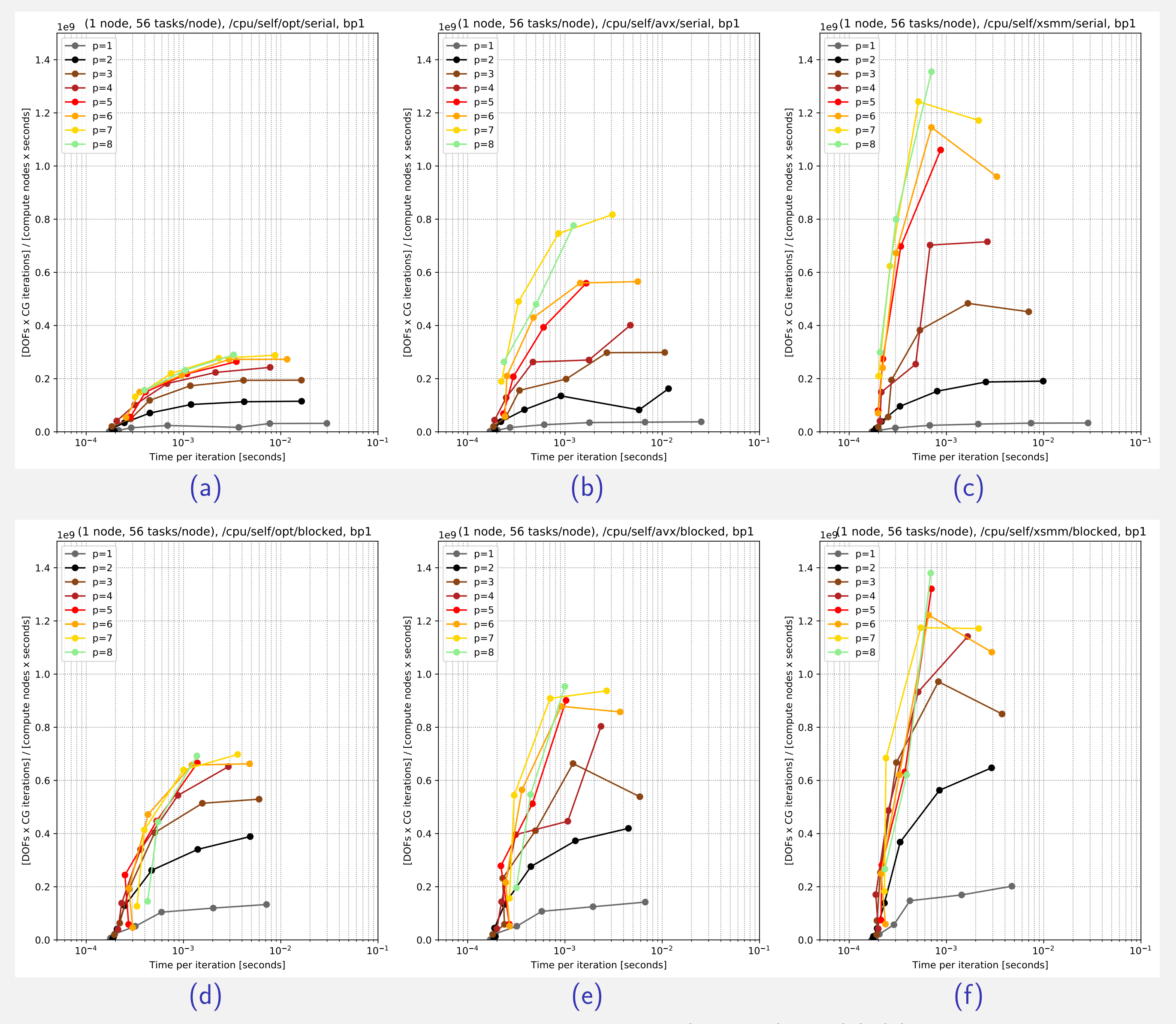

Figure 4: BP1: 2x Intel Xeon Platinum 8180M CPU 2.50GHz (Skylake): In (a)-(c), serial implementation. In (d)-(f) blocked implementation.

## Navier-Stokes Demo

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

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

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

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

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 $\mu$ the dynamic viscosity coefficient, and $\lambda = -2/3$ the Stokes hypothesis constant. In equations (1), $\rho$ 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 $3 \times 3$ 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). We solve the density current problem: a cold air bubble drops by convection in a neutrally stratified atmosphere.

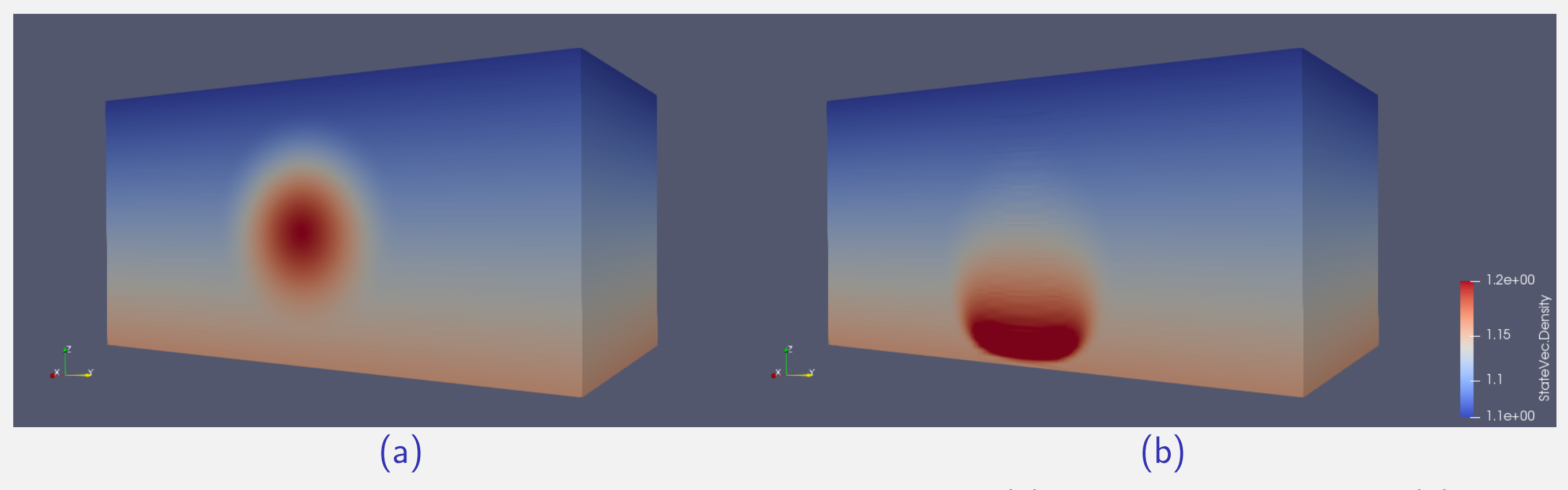

Figure 5: Solution of the compressible Navier-Stokes equations. In (a) the initial condition; In (b) the evolution at time $t = 50$ s.

## Outlook

Status: ready for collaborators and friendly users. Ongoing work on CUDA and HIP optimizations.

- Improved non-conforming and mixed-mesh support
- Algorithmic differentiation of Q-functions

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