# Efficient high-order operators and library reuse: libCEED

Valeria Barra, **Jed Brown**, **Jeremy Thompson**

CU Boulder (CS)

LANS seminar, ANL, June 26, 2019

CEED
EXASCALE DISCRETIZATIONS

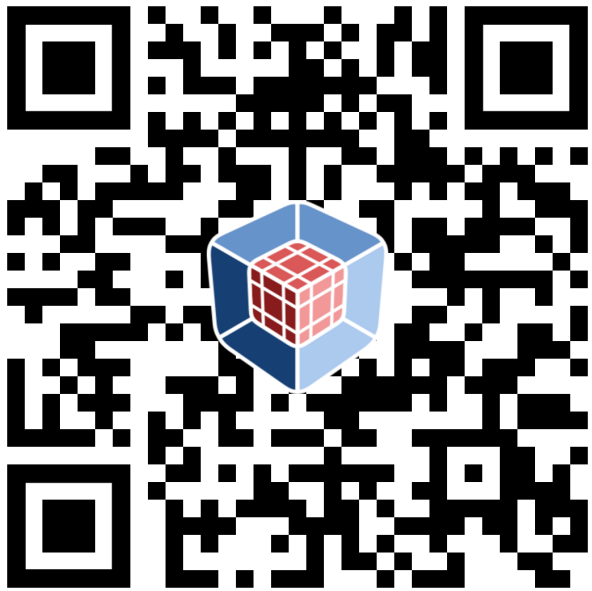
# Overview

- For decades, high-order numerical methods have been considered too expensive
- A sparse matrix is no longer a good representation for high-order operators. In particular, the Jacobian of a nonlinear operator is known to rapidly lose sparsity as the order is increased
- libCEED uses a matrix-free operator description, based on a purely algebraic interface, where user only specifies action of weak form operators
- libCEED operator representation is optimal with respect to the FLOPs needed for its evaluation, as well as the memory transfer needed for operator evaluations (matvec)
  - Matrix-free operators that exploit tensor-product structures reduce the work load from $O(p^6)$ (for sparse matrix) to $O(p^4)$, and memory storage from $O(p^6)$ to $O(p^3)$
- We demonstrate the usage of libCEED with PETSc for a compressible Navier Stokes solver

University of Colorado Boulder

# libCEED: the library of the CEED (Center for Efficient Extensible Discretizations)

- Primary target: high order finite element methods (FEM) exploiting tensor product structure
- Open source (BSD-2 license) C library with Fortran interface
- Releases: v0.1 (January 2018), v0.2 (March 2018), v.0.3 (September 2018), v0.4 (March 2019)



For latest release:
Brown J., Abdelfattah A., Barra V., Dobrev V., Dudouit Y., Fischer P., et al., *CEED ECP Milestone Report: Public release of CEED 2.0* (2019, March 31) DOI: `http://doi.org/10.5281/zenodo.2641316`

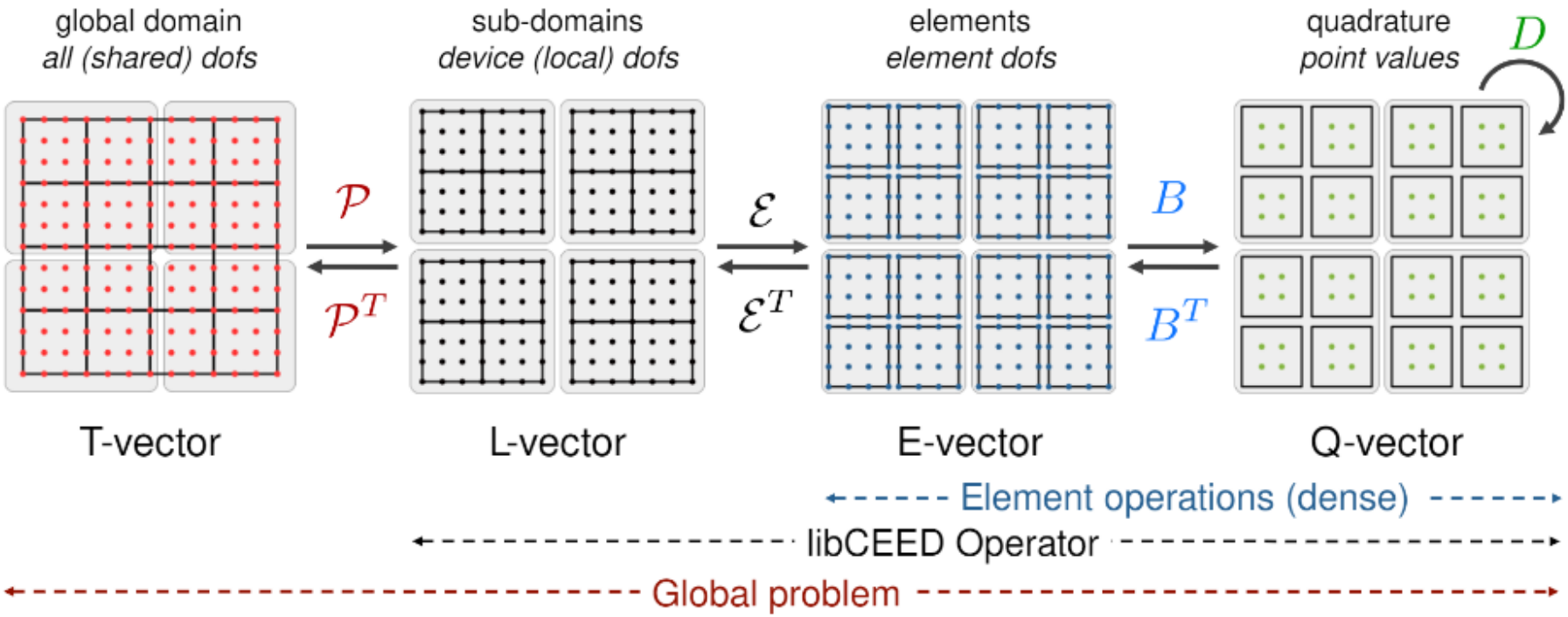
University of Colorado Boulder

# libCEED: the library of the CEED (Center for Efficient Extensible Discretizations)

- Extensible backends:
  - CPU: reference and vectorized implementations
  - OCCA (just-in-time compilation) for CPUs, GPUs, OpenMP (Open Multi-Processing: API that supports multi-platform shared memory programming), OpenCL (framework for writing programs that execute across several platforms, e.g., CPUs, GPUs, etc.)
  - MAGMA (dense Linear Algebra library for GPUs and Multicore Architectures)
  - CUDA (parallel computing platform and API for general purpose processing on GPUs)
  - AVX (Advanced Vector Extensions instruction set architecture extension) and LIBXSMM (Intel library for small dense and sparse matrix multiplications)
- Same source code can call multiple CEEDs with different backends. On-device operator implementation with unique interface

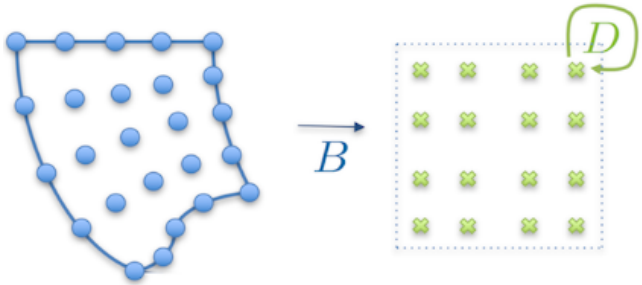
# libCEED decomposition

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



# libCEED API objects

- $\mathcal{E}$: Ceed Element Restriction
  - Restrict to single element
  - User choice in ordering




  - $\mathrm{B}$: Ceed Basis Applicator
    - Describes the actions on basis such as interpolation, gradient, div, curl
    - Independent of geometry and element topology
  - $\mathrm{D}$: Ceed QFunction
    - Operator that defines the action of the physics at quadrature points
    - Choice of interlaced (by fields) or blocked (by element) for multi-component vectors
  - $\mathrm{C} = \mathcal{E}^{\mathcal{T}}\mathrm{B}^{\mathsf{T}}\mathrm{D}\mathrm{B}\mathcal{E}$: CeedOperator
    - Composition of different operators defined on different element topologies possible
  - $\mathrm{A} = \mathrm{P}^{\mathsf{T}}\mathrm{C}\mathrm{P}$: User code responsible for parallelization on different compute devices. We use PETSc

University of Colorado Boulder

# Composition of solvers for multiphysics problems

The algebraic system obtained by the discretization of an $m$-variable nonlinear PDE is $\mathbf{F}(\mathbf{u}) = \mathbf{0}$:

$$\begin{pmatrix} F_1(u_1, u_2, \ldots, u_m) \\ F_2(u_1, u_2, \ldots, u_m) \\ \vdots \\ F_m(u_1, u_2, \ldots, u_m) \end{pmatrix} = \mathbf{0} \quad \xrightarrow{\text{Jacobian}} \quad \begin{pmatrix} J_{11} & J_{12} & \cdots & J_{1m} \\ J_{21} & J_{22} & \cdots & J_{2m} \\ \vdots & & \ddots & \\ J_{m1} & J_{m2} & \cdots & J_{mm} \end{pmatrix}$$

solved via Newton's method: $\mathbf{u}^{n+1} = \mathbf{u}^n - \lambda \hat{\mathbf{J}}^{-1}(\mathbf{u}^n)\mathbf{F}(\mathbf{u}^n)$.

# Composition of solvers for multiphysics problems

The algebraic system obtained by the discretization of an $m$-variable nonlinear PDE is $\mathbf{F}(\mathbf{u}) = \mathbf{0}$:

$$\begin{pmatrix} F_1(u_1, u_2, \ldots, u_m) \\ F_2(u_1, u_2, \ldots, u_m) \\ \vdots \\ F_m(u_1, u_2, \ldots, u_m) \end{pmatrix} = \mathbf{0} \quad \xrightarrow{\text{Jacobian}} \quad \begin{pmatrix} J_{11} & J_{12} & \cdots & J_{1m} \\ J_{21} & J_{22} & \cdots & J_{2m} \\ \vdots & & \ddots & \\ J_{m1} & J_{m2} & \cdots & J_{mm} \end{pmatrix}$$

solved via Newton's method: $\mathbf{u}^{n+1} = \mathbf{u}^n - \lambda \hat{J}^{-1}(\mathbf{u}^n)\mathbf{F}(\mathbf{u}^n)$.

Ex: For a Dirichlet Stokes flow

$$\begin{array}{rcl} \nabla \cdot \boldsymbol{\sigma} & = & \mathbf{F_b} \\ \nabla \cdot \mathbf{u} & = & 0 \end{array} \Rightarrow \begin{pmatrix} J_{uu} & J_{pu}^T \\ J_{pu} & 0 \end{pmatrix}$$

where $\boldsymbol{\sigma} = \mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T) - p\mathbf{I}_3$, and where the Schur's complement is $S = -J_{pu} J_{uu}^{-1} J_{pu}^T$ (needs preconditioning). If we use the simple block Jacobi preconditioner $\rightarrow$ block Gauss-Seidel, that can be solved by only partial assembly of the Jacobian, where each block can be computed independently and we can reuse the same QFunction for the blocks corresponding to different physical variables.

University of Colorado Boulder

# libCEED API for operator composition

Creation of QFunctions and CEED operators:

```
CeedQFunctionCreateInterior(ceed, 1, Mass, __FILE__":Mass", &qf_mass);
CeedQFunctionAddInput(qf_mass, "u", 5, CEED_EVAL_INTERP);
CeedQFunctionAddInput(qf_mass, "weights", 1, CEED_EVAL_NONE);
CeedQFunctionAddOutput(qf_mass, "v", 5, CEED_EVAL_INTERP);

CeedOperatorCreate(ceed, qf_mass, NULL, NULL, &op_mass);
CeedOperatorSetField(op_mass, "u", Erestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Erestrictudi, CEED_NOTRANSPOSE, basisx, weights);
CeedOperatorSetField(op_mass, "v", Erestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
```

# libCEED API for operator composition

Creation of QFunctions and CEED operators:

```
CeedQFunctionCreateInterior(ceed, 1, Mass, __FILE__":Mass", &qf_mass);
CeedQFunctionAddInput(qf_mass, "u", 5, CEED_EVAL_INTERP);
CeedQFunctionAddInput(qf_mass, "weights", 1, CEED_EVAL_NONE);
CeedQFunctionAddOutput(qf_mass, "v", 5, CEED_EVAL_INTERP);

CeedOperatorCreate(ceed, qf_mass, NULL, NULL, &op_mass);
CeedOperatorSetField(op_mass, "u", Erestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Erestrictudi, CEED_NOTRANSPOSE, basisx, weights);
CeedOperatorSetField(op_mass, "v", Erestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
```

$$f(\mathbf{u}, \mathbf{p}; \Theta) \rightleftharpoons f(u, p; \mathbf{\Theta})$$

University of Colorado Boulder

# libCEED API for operator composition

Creation of QFunctions and CEED operators:

```
CeedQFunctionCreateInterior(ceed, 1, Mass, __FILE__":Mass", &qf_mass);
CeedQFunctionAddInput(qf_mass, "u", 5, CEED_EVAL_INTERP);
CeedQFunctionAddInput(qf_mass, "weights", 1, CEED_EVAL_NONE);
CeedQFunctionAddOutput(qf_mass, "v", 5, CEED_EVAL_INTERP);

CeedOperatorCreate(ceed, qf_mass, NULL, NULL, &op_mass);
CeedOperatorSetField(op_mass, "u", Erestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Erestrictudi, CEED_NOTRANSPOSE, basisx, weights);
CeedOperatorSetField(op_mass, "v", Erestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
```

$$f(\mathbf{u}, \mathbf{p}; \Theta) \rightleftharpoons f(u, p; \mathbf{\Theta})$$

Composition of operators for multiphysics or hybrid-element meshes:

```
CeedCompositeOperatorCreate(ceed, &op_comp);
CeedCompositeOperatorAddSub(op_comp, op_1);
CeedCompositeOperatorAddSub(op_comp, op_2);
```

University of Colorado Boulder

# Performance w. r. t. space on KNL

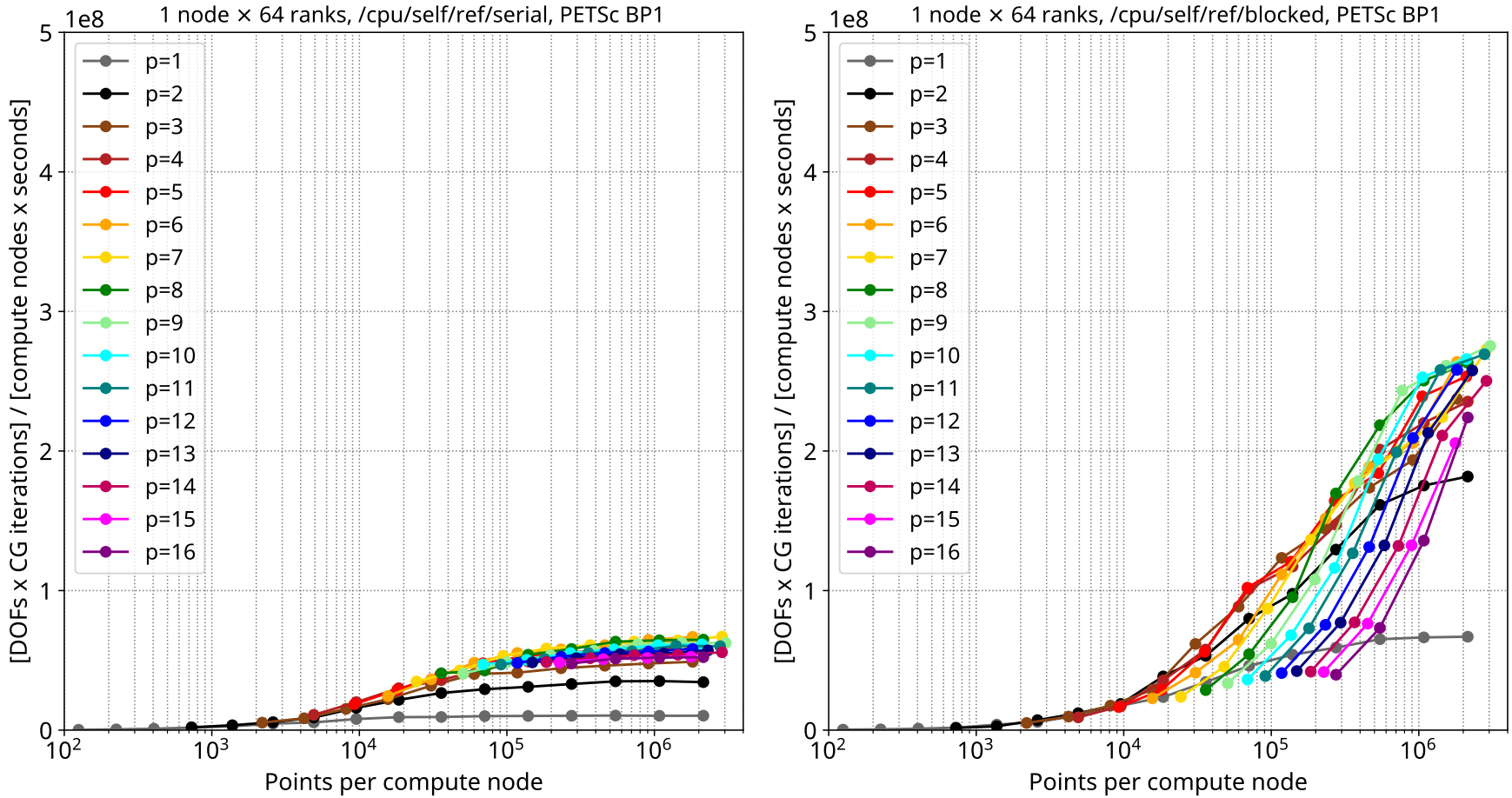
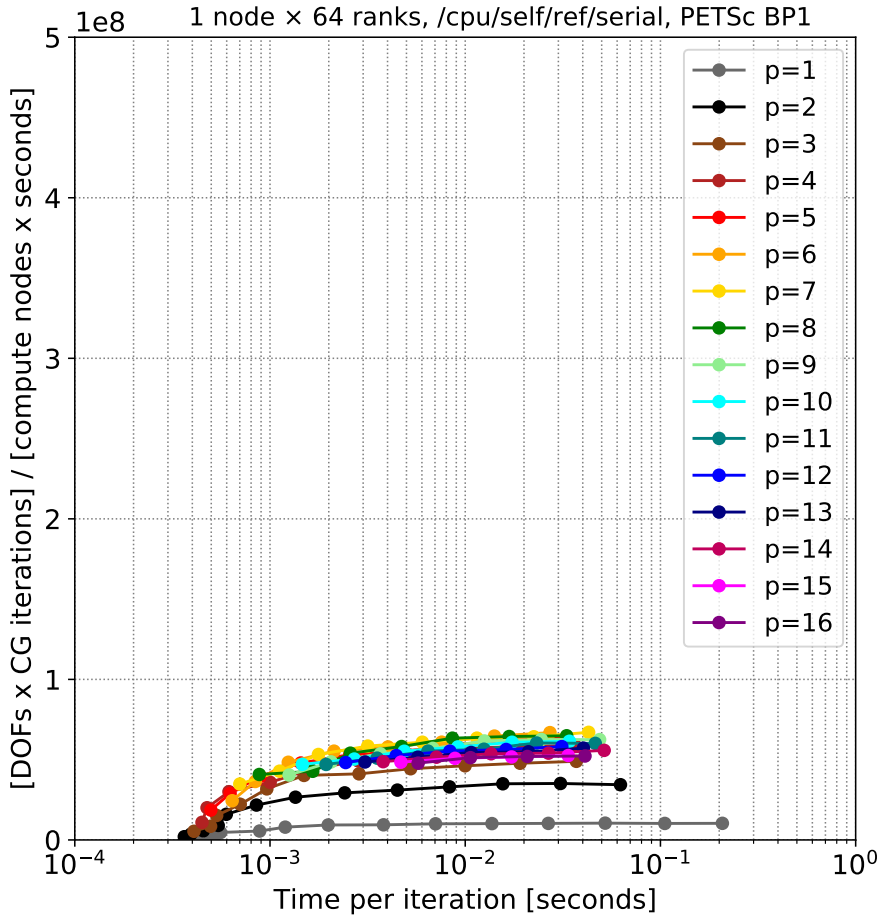


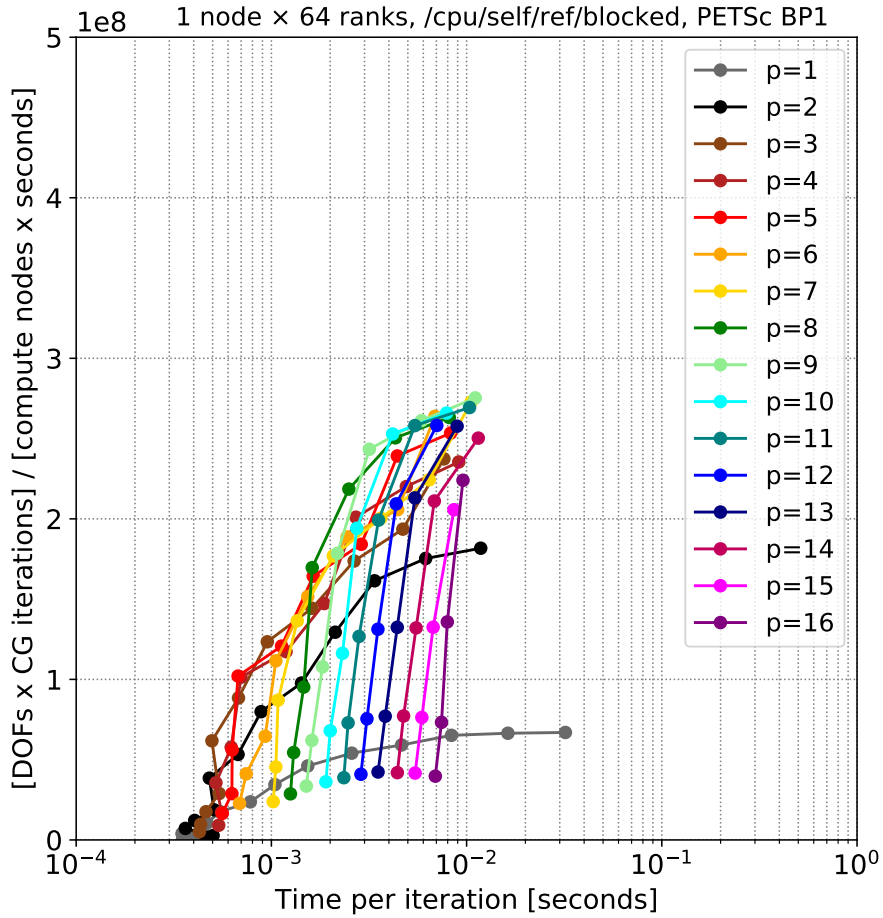

Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz) with intel-18 compiler. In (a) serial implementation; in (b) blocked implementation ($q = P + 1$, $P = p + 1$)

University of Colorado Boulder

# Performance w. r. t. time on KNL




(a)

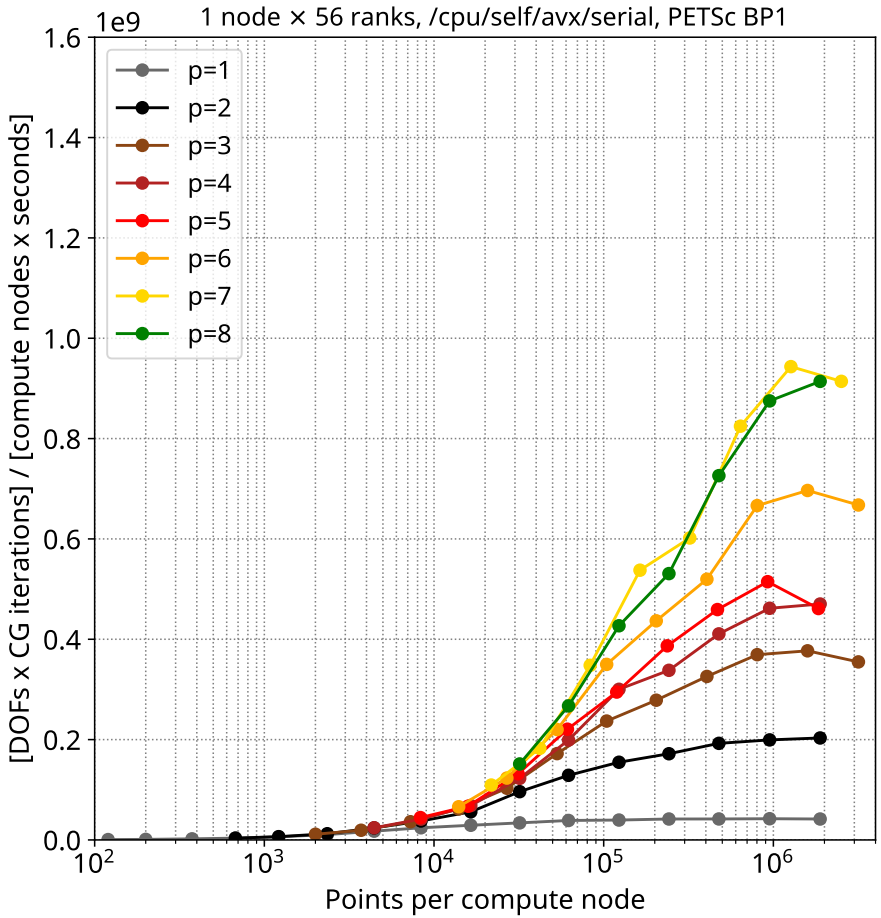



(b)

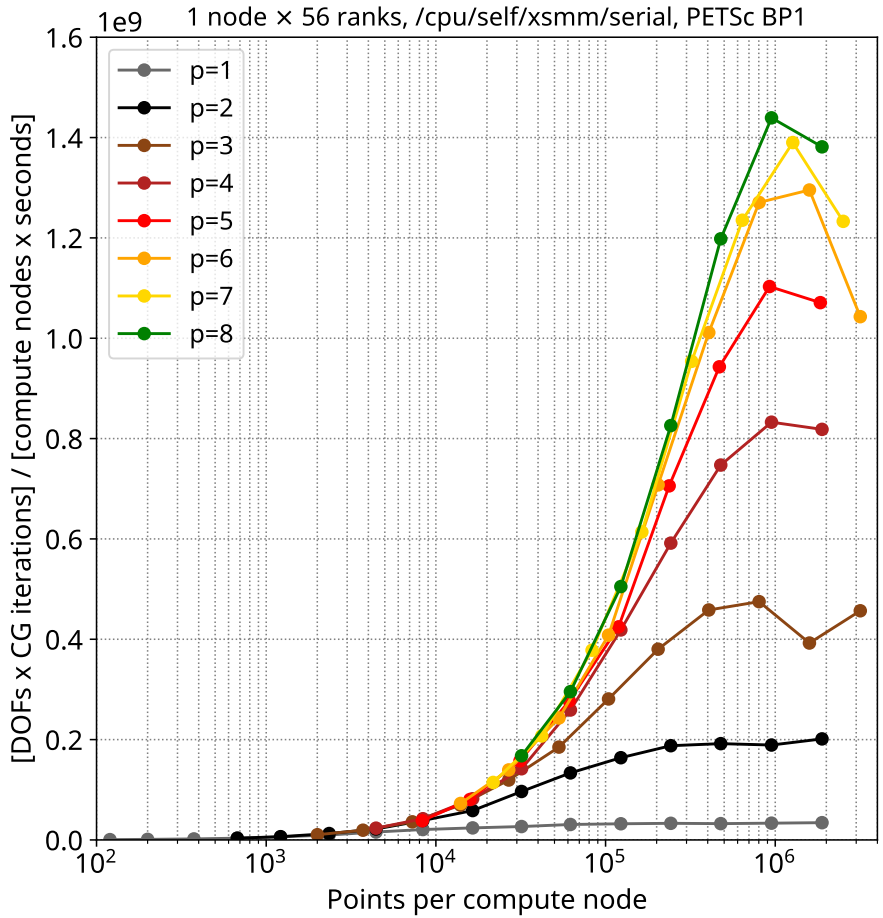
Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz) with intel-18 compiler. In (a) serial implementation; in (b) blocked implementation ($q = P + 1$, $P = p + 1$)

University of Colorado Boulder

# Performance w. r. t. space on Skylake: serial




(a)

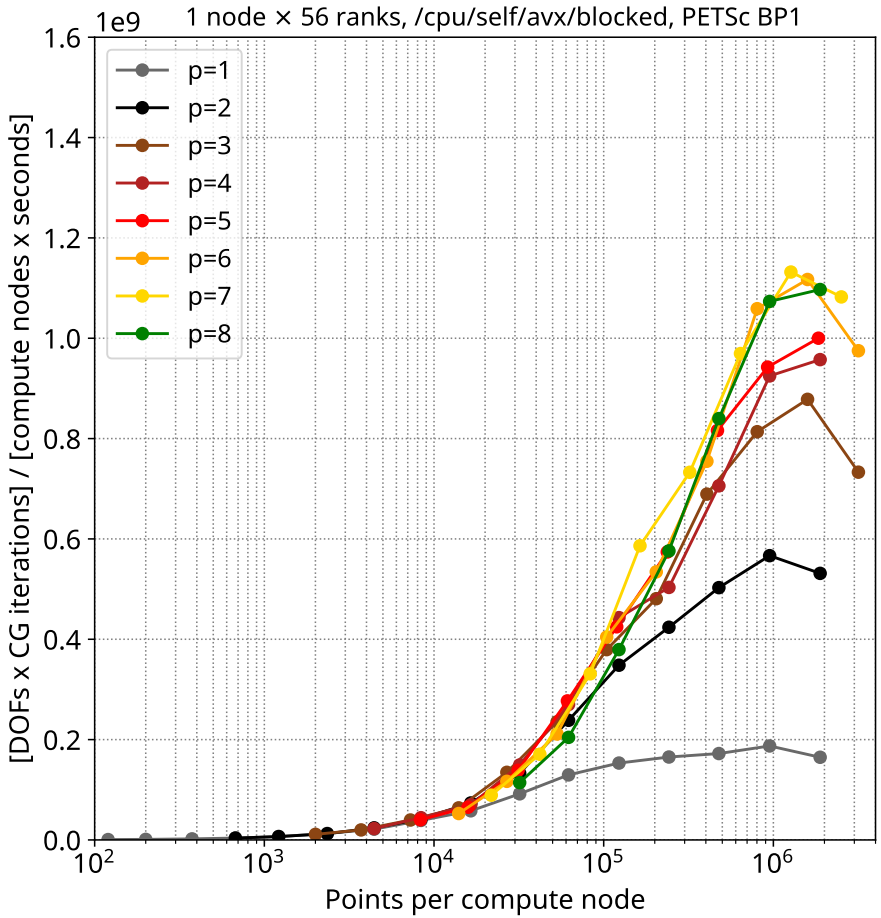



(b)

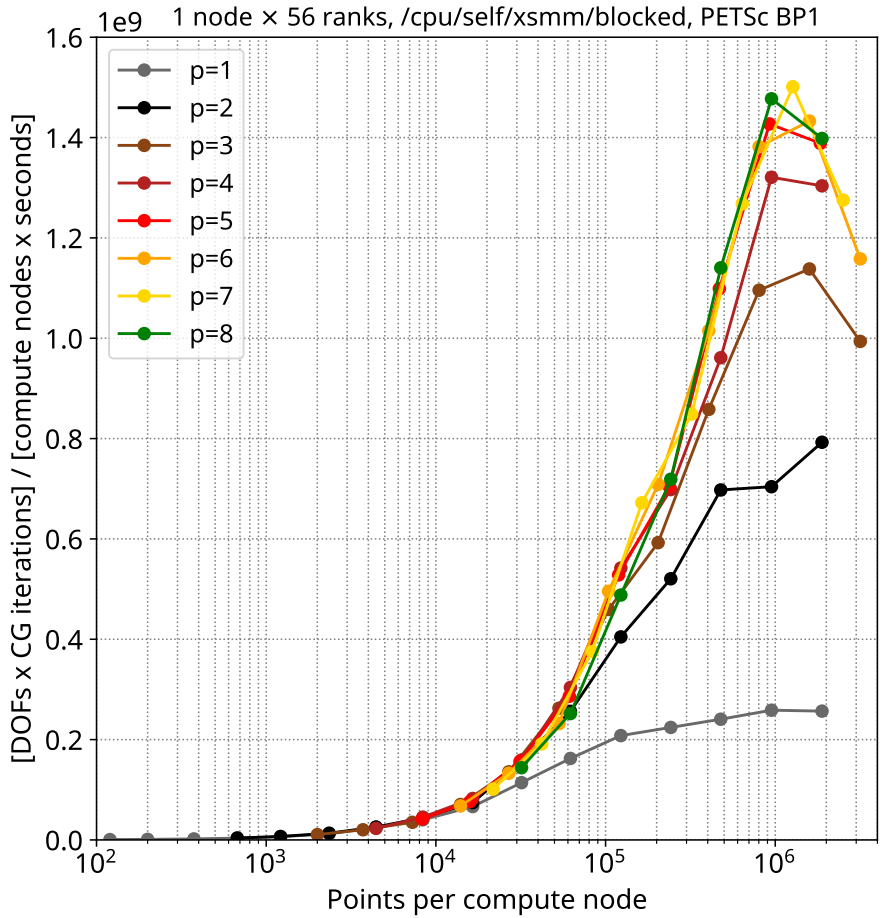
Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with intel-19 compiler. Backends: in (a) AVX; in (b) libXSMM ($q = P + 1$, $P = p + 1$)

University of Colorado Boulder

# Performance w. r. t. space on Skylake: blocked




(a)

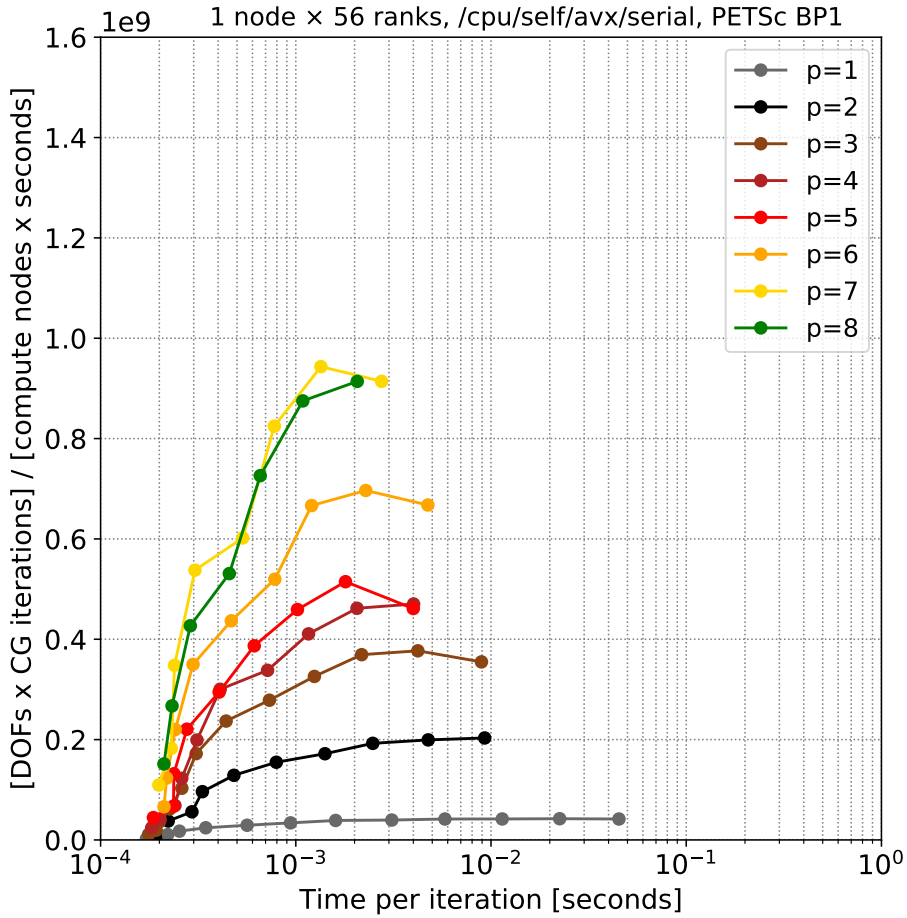



(b)

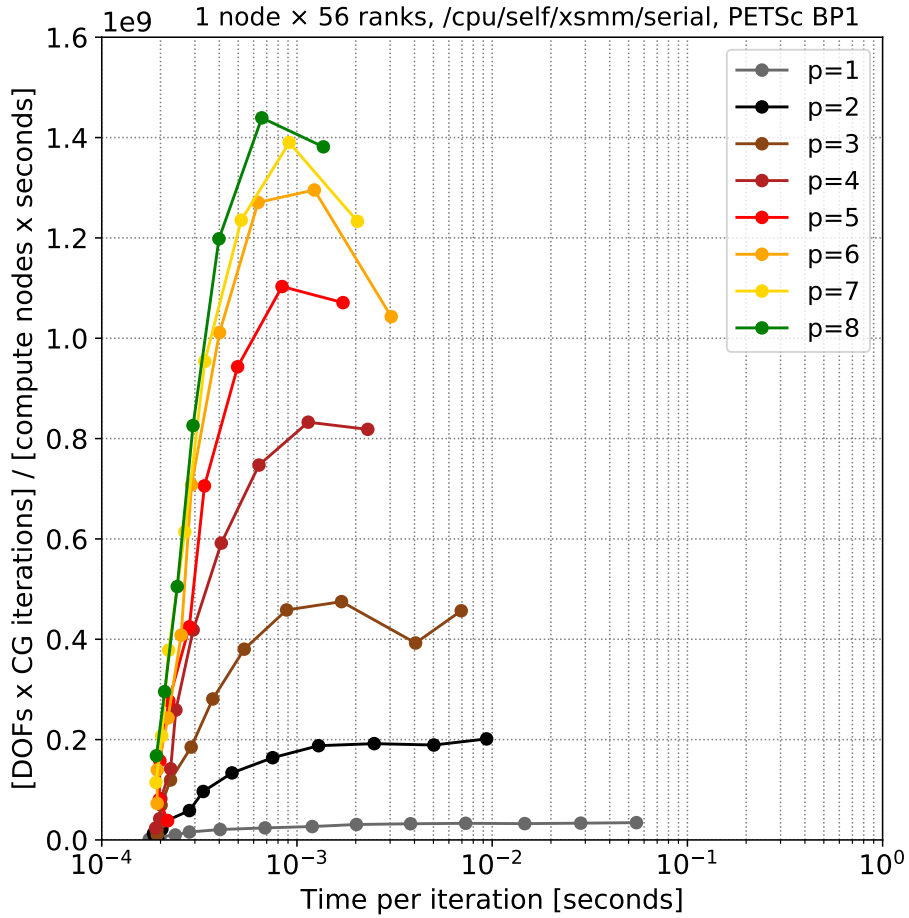
Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with intel-19 compiler. Backends: in (a) AVX; in (b) libXSMM ($q = P + 1$, $P = p + 1$)

University of Colorado Boulder

# Performance w. r. t. time on Skylake: serial




(a)

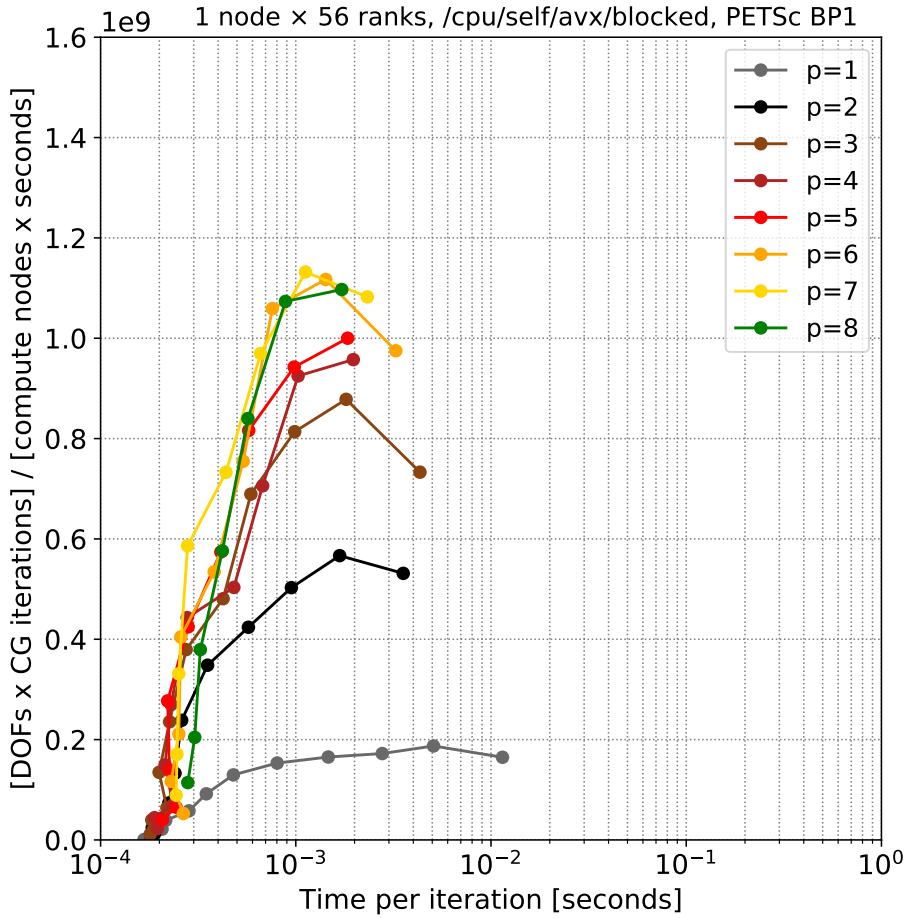



(b)

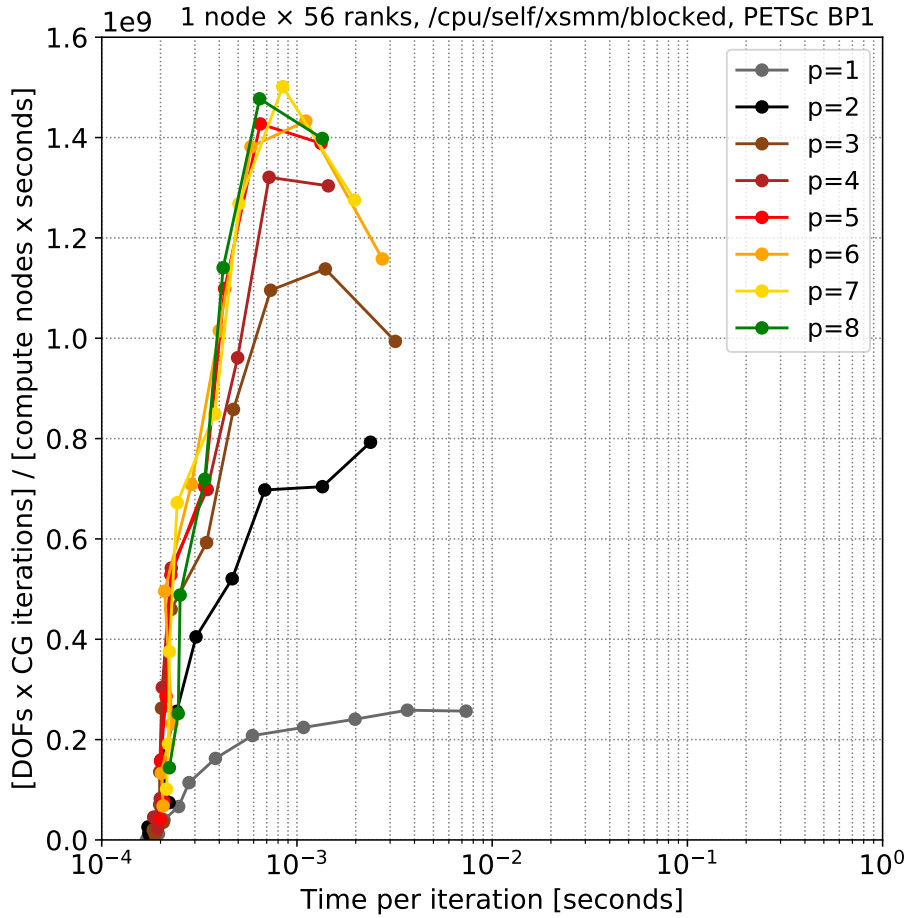
Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with intel-19 compiler.
Backends: in (a) AVX; in (b) libXSMM ($q = P + 1$, $P = p + 1$)

University of Colorado Boulder

# Performance w. r. t. time on Skylake: blocked




(a)




(b)

Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with intel-19 compiler.
Backends: in (a) AVX; in (b) libXSMM ($q = P + 1$, $P = p + 1$)

University of Colorado Boulder

# Towards a libCEED miniapp: a Navier-Stokes solver

Compressible Navier-Stokes equations in conservation form:

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

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

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

# Towards a libCEED miniapp: a Navier-Stokes solver

Compressible Navier-Stokes equations in conservation form:

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

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

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

where $\boldsymbol{\sigma} = \mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T + \lambda(\nabla \cdot \mathbf{u})\mathbf{I}_3)$, and

# Towards a libCEED miniapp: a Navier-Stokes solver

Compressible Navier-Stokes equations in conservation form:

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

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

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

where $\boldsymbol{\sigma} = \mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T + \lambda(\nabla \cdot \mathbf{u})\mathbf{I}_3)$, and

$$(c_p/c_v - 1)\,(E - \mathbf{U} \cdot \mathbf{U}/(2\rho) - \rho g z) = P \quad \leftarrow \text{ pressure}$$

$\mu \leftarrow$ dynamic viscosity

$g \leftarrow$ gravitational acceleration

$k \leftarrow$ thermal conductivity

$\lambda \leftarrow$ Stokes hypothesis constant

$c_p \leftarrow$ specific heat, constant pressure

$c_v \leftarrow$ specific heat, constant volume

University of Colorado Boulder

# Vector form

The system (1) can be rewritten in vector form

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = S(\mathbf{q}), \tag{2}$$

for the state variables

$$\mathbf{q} = \begin{pmatrix} \rho \\ \mathbf{U} \equiv \rho \mathbf{u} \\ E \equiv \rho e \end{pmatrix} \begin{matrix} \\ \leftarrow \text{ momentum density} \\ \leftarrow \text{ energy density} \end{matrix} \tag{3}$$

# Vector form

The system (1) can be rewritten in vector form

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = S(\mathbf{q}), \tag{2}$$

for the state variables

$$\mathbf{q} = \begin{pmatrix} \rho \\ \mathbf{U} \equiv \rho \mathbf{u} \\ E \equiv \rho e \end{pmatrix} \begin{matrix} \leftarrow \text{ volume mass density} \\ \leftarrow \text{ momentum density} \\ \leftarrow \text{ energy density} \end{matrix} \tag{3}$$

where

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

$$S(\mathbf{q}) = -\begin{pmatrix} 0 \\ \rho g \hat{\mathbf{k}} \\ 0 \end{pmatrix}$$

# Space discretization

We use high-order finite elements/spectral elements: high-order Lagrange polynomials over non-uniformly spaced nodes, the Legendre-Gauss-Lobatto (LGL) points (roots of the $p^{th}$-order Legendre polynomial $P_p$). We let $\mathbb{R}^3 \supset \Omega = \bigcup_{e=1}^{N_e} \Omega_e$, with $N_e$ disjoint hexaedral elements.

The physical coordinates are $\mathbf{x} = (x, y, z) \in \Omega_e$, while the reference coords are $\boldsymbol{\xi} = (\xi, \eta, \zeta) \in \mathbf{I} = [-1, 1]^3$.

# Space discretization

We use high-order finite elements/spectral elements: high-order Lagrange polynomials over non-uniformly spaced nodes, the Legendre-Gauss-Lobatto (LGL) points (roots of the $p^{th}$-order Legendre polynomial $P_p$). We let $\mathbb{R}^3 \supset \Omega = \bigcup_{e=1}^{N_e} \Omega_e$, with $N_e$ disjoint hexaedral elements.

The physical coordinates are $\mathbf{x} = (x, y, z) \in \Omega_e$, while the reference coords are $\boldsymbol{\xi} = (\xi, \eta, \zeta) \in \mathbf{I} = [-1, 1]^3$.

Define the discrete solution

$$\mathbf{q}_N(\mathbf{x}, t)^{(e)} = \sum_{k=1}^{P} \psi_k(\mathbf{x}) \mathbf{q}_k^{(e)} \tag{4}$$

with $P$ the number of nodes in the element $(e)$.

We use tensor-product bases $\psi_{kji} = h_i(\xi) h_j(\eta) h_k(\zeta)$.

# Strong and weak formulations

The strong form of (3):

$$\int_\Omega \nu \left( \frac{\partial \mathbf{q}_N}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}_N) \right) \, d\Omega = \int_\Omega \nu \mathbf{S}(\mathbf{q}_N) \, d\Omega \, , \; \forall \nu \in \mathcal{V}_p \tag{5}$$

with $\mathcal{V}_p = \{\nu \in H^1(\Omega_e) \, | \, \nu \in P_p(\mathbf{I}), e = 1, \ldots, N_e\}$.
Weak form:

$$\int_\Omega \nu \frac{\partial \mathbf{q}_N}{\partial t} \, d\Omega + \int_\Gamma \nu \hat{\mathbf{n}} \cdot \mathbf{F}(\mathbf{q}_N) \, d\Omega - \int_\Omega \nabla \nu \cdot \mathbf{F}(\mathbf{q}_N) \, d\Omega = \int_\Omega \nu \mathbf{S}(\mathbf{q}_N) \, d\Omega \, , \; \forall \nu \in \mathcal{V}_p \tag{6}$$

# Strong and weak formulations

The strong form of (3):

$$\int_\Omega \nu \left( \frac{\partial \mathbf{q}_N}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}_N) \right) \, d\Omega = \int_\Omega \nu \mathbf{S}(\mathbf{q}_N) \, d\Omega \,, \; \forall \nu \in \mathcal{V}_p \tag{5}$$

with $\mathcal{V}_p = \{\nu \in H^1(\Omega_e) \,|\, \nu \in P_p(\mathbf{I}), e = 1, \ldots, N_e\}$.
Weak form:

$$\int_\Omega \nu \frac{\partial \mathbf{q}_N}{\partial t} \, d\Omega + \int_\Gamma \nu \hat{\mathbf{n}} \cdot \mathbf{F}(\mathbf{q}_N) \, d\Omega - \int_\Omega \nabla \nu \cdot \mathbf{F}(\mathbf{q}_N) \, d\Omega = \int_\Omega \nu \mathbf{S}(\mathbf{q}_N) \, d\Omega \,, \; \forall \nu \in \mathcal{V}_p \tag{6}$$

For the Time Discretization we use an explicit formulation

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

solved with the adaptive Runge-Kutta-Fehlberg (RKF4-5) method

University of Colorado Boulder

# A very simple example: The advection equation

We analyze the transport of total energy

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

with $\mathbf{u}$ a uniform circular motion. BCs: no-slip and non-penetration for $\mathbf{u}$, no-flux for $E$.

order:
$p = 6$
$\Omega =$
$[0, 2000]^3$ m
elem.
resolution:
250 m
Nodes: 117 649

# Top view: Advection

# Application example: Density current

A cold air bubble drops by convection in a neutrally stratified atmosphere.

Its initial condition is defined in terms of the Exner pressure, $\pi(\mathbf{x}, t)$, and potential temperature, $\theta(\mathbf{x}, t)$, that relate to the state variables via

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

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

where $P_0$ is the atmospheric pressure.

BCs: no-slip and non-penetration for $\mathbf{u}$, no-flux for mass and energy densities.

University of Colorado Boulder

# Density current

order: $\mathrm{p} = 8$, $\Omega = [0, 6000]^2\ \mathrm{m} \times [0, 3000]\ \mathrm{m}$, elem. resolution: 125 m,
Nodes:
1 081 665

University of Colorado Boulder

# Application example: Wind turbine

We simulate the aerodynamics of a wind turbine through an Actuator Disc Model (ADM). ADM models the turbine as a disc, with a uniform thrust force

$$F_T = \frac{1}{2}\rho u_{1,\infty}^2 A_D C_T \,,$$

$\rho$ $\leftarrow$ density of air
$u_{1,\infty}$ $\leftarrow$ unperturbed (far field) axial velocity
$A_D$ $\leftarrow$ area swept by rotor
$C_T$ $\leftarrow$ thrust coefficient

# Application example: Wind turbine

We simulate the aerodynamics of a wind turbine through an Actuator Disc Model (ADM). ADM models the turbine as a disc, with a uniform thrust force

$$F_T = \frac{1}{2}\rho u_{1,\infty}^2 A_D C_T \,,$$

$$\begin{aligned} \rho &\leftarrow \text{density of air} \\ u_{1,\infty} &\leftarrow \text{unperturbed (far field) axial velocity} \\ A_D &\leftarrow \text{area swept by rotor} \\ C_T &\leftarrow \text{thrust coefficient} \end{aligned}$$

We project this nodal force onto the surrounding cells via

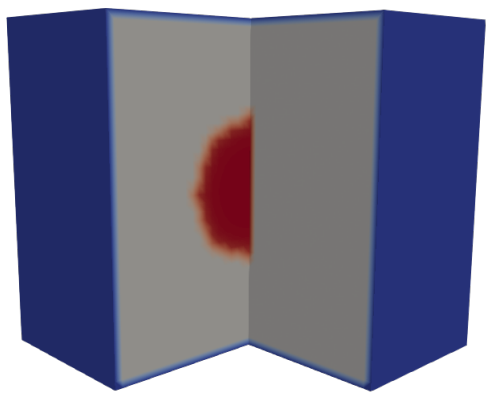
$$F_{turbine} = F_T f_\varepsilon \,, \quad \text{where } f_\varepsilon = \frac{e^{-(r/\varepsilon)^2}}{\varepsilon^3 \pi^{3/2}}$$

# Application example: Wind turbine

We simulate the aerodynamics of a wind turbine through an Actuator Disc Model (ADM). ADM models the turbine as a disc, with a uniform thrust force

$$F_T = \frac{1}{2}\rho u_{1,\infty}^2 A_D C_T\,,$$

$$\begin{aligned} \rho &\leftarrow \text{density of air} \\ u_{1,\infty} &\leftarrow \text{unperturbed (far field) axial velocity} \\ A_D &\leftarrow \text{area swept by rotor} \\ C_T &\leftarrow \text{thrust coefficient} \end{aligned}$$



We project this nodal force onto the surrounding cells via

$$F_{turbine} = F_T f_\varepsilon\,, \quad \text{where } f_\varepsilon = \frac{e^{-(r/\varepsilon)^2}}{\varepsilon^3\pi^{3/2}}$$

This adds a source/sink for momentum in the conservation equation $\frac{\partial q}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = S(\mathbf{q})$, with $S(\mathbf{q}) = \left(0, F_{turbine}\hat{\mathbf{i}} - \rho g\hat{\mathbf{k}}, 0\right)$.

University of Colorado Boulder

# Conclusions and outlook

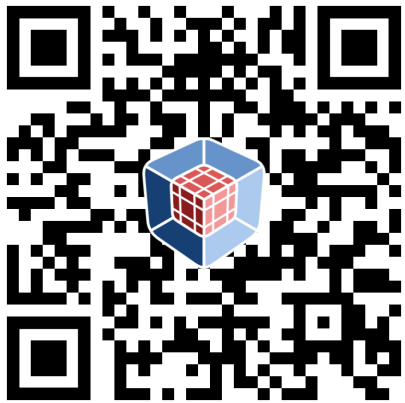
- We have demonstrated the use of libCEED with PETSc for the numerical high-order solutions of
  - Advection equation
  - Full compressible Navier-Stokes equations
  - Actuator Disc Model for wind turbines
- libCEED always welcomes contributors and friendly users

  `https://github.com/CEED/libCEED`

University of Colorado Boulder

# Conclusions and outlook

- We have demonstrated the use of libCEED with PETSc for the numerical high-order solutions of
  - Advection equation
  - Full compressible Navier-Stokes equations
  - Actuator Disc Model for wind turbines
- libCEED always welcomes contributors and friendly users
  `https://github.com/CEED/libCEED`



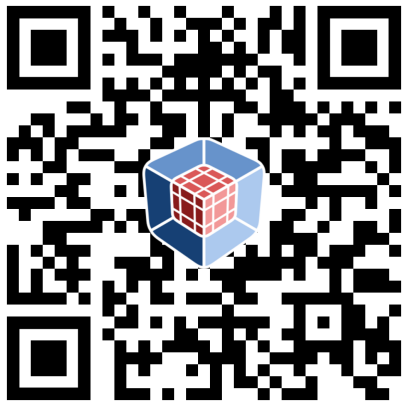
Ongoing and future work:

- Actuator Line or Surface Models to represent wake dynamics of wind turbines with moving meshes
- Improved non-conforming and mixed-mesh support
- Implicit-explicit (IMEX) and fully implicit time discretizations
- Algorithmic differentiation of Q-functions
- Ongoing work on CUDA and HIP optimizations.

University of Colorado Boulder

# Conclusions and outlook

- We have demonstrated the use of libCEED with PETSc for the numerical high-order solutions of
  - Advection equation
  - Full compressible Navier-Stokes equations
  - Actuator Disc Model for wind turbines
- libCEED always welcomes contributors and friendly users
  `https://github.com/CEED/libCEED`



Ongoing and future work:

- Actuator Line or Surface Models to represent wake dynamics of wind turbines with moving meshes
- Improved non-conforming and mixed-mesh support
- Implicit-explicit (IMEX) and fully implicit time discretizations
- Algorithmic differentiation of Q-functions
- Ongoing work on CUDA and HIP optimizations.

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

University of Colorado Boulder