

libCEED: a versatile, extensible high-order finite elements library for performance portability

Valeria Barra¹, Jed Brown¹, Jeremy Thompson^{1,2}, Yohann Dudouit³

¹ Department of Computer Science, CU Boulder

² Department of Applied Math, CU Boulder

³ Lawrence Livermore National Lab

Applied Math Seminar
Colorado State University
October 10, 2019

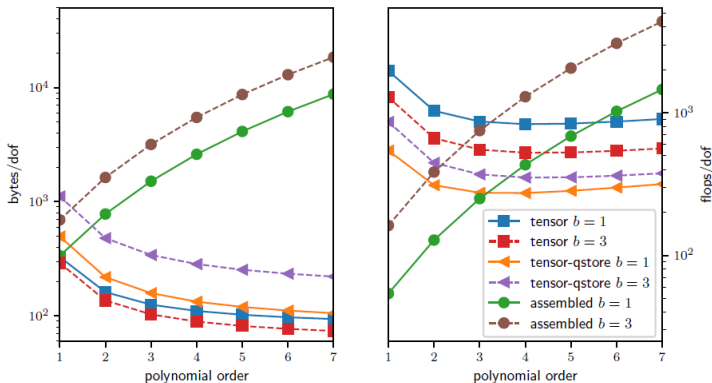


CEED
EXASCALE DISCRETIZATIONS



EXASCALE COMPUTING PROJECT

Motivation: Why matrix-free? And why high-order?



Memory bandwidth (left) and flops per dof (right) to apply a Jacobian matrix, obtained from discretizations of a b -variable PDE system. Assembled matrix vs matrix-free (exploits the tensor product structure by either storing at q -points or computing on the fly)

[Courtesy: Jed Brown]

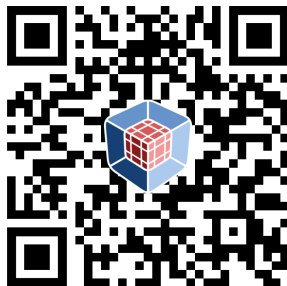
Overview

- 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 some application examples



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

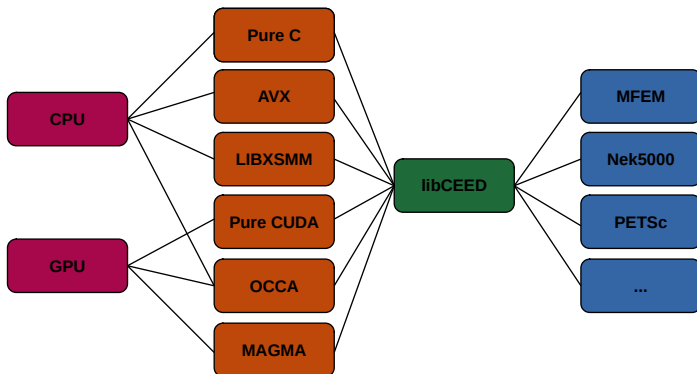
- Primary target: high-order finite/spectral element methods (FEM/SEM) exploiting tensor-product structure
- Open source (BSD-2 license) C library with Fortran interface
- Releases: v0.1 (January 2018), v0.2 (March 2018), v.0.3 (September 2018), v0.4 (March 2019), v0.5 (September 2019)



For latest release:

Tomov S., Abdelfattah A., **Barra V.**, Beams N., Brown J. et al., *CEED ECP Milestone Report: Performance tuning of CEED software and 1st and 2nd wave apps* (2019, Oct 2nd) DOI: <https://doi.org/10.5281/zenodo.3477618>

libCEED backends



libCEED backends

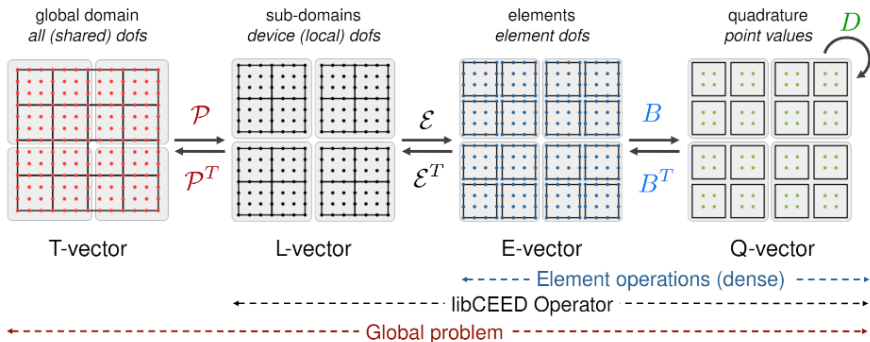
<code>/cpu/self/ref/*:</code>	with <code>*</code> reference serial and blocked implementations
<code>/cpu/self/avx/*:</code>	AVX (Advanced Vector Extensions instruction sets) with <code>*</code> reference serial and blocked implementations
<code>/cpu/self/xsmm/*:</code>	LIBXSMM (Intel library for small dense/sparse mat-multiply) with <code>*</code> reference serial and blocked implementations
<code>/*/occa:</code>	OCCA (just-in-time compilation) with <code>*</code> : CPU, GPU, OpenMP (Open Multi-Processing: API), OpenCL (framework for CPUs, GPUs, etc.)
<code>/gpu/magma:</code>	CUDA MAGMA (dense Linear Algebra library for GPUs and multicore architectures) kernels
<code>/gpu/cuda/*:</code>	CUDA with <code>*</code> : ref (reference pure CUDA kernels), reg (CUDA kernels using one thread per element), shared , optimized CUDA kernels using shared memory gen , optimized CUDA kernels using code generation

Same source code can call multiple CEEDs with different backends. On-device operator implementation with unique interface

libCEED decomposition



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

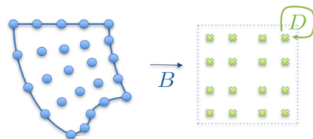


libCEED API objects

- $\mathcal{E}$: Ceed Element Restriction

Restrict to single element

User choice in ordering



- B : Ceed Basis Applicator

Describes the actions on basis such as interpolation, gradient, div, curl

Independent of geometry and element topology

- 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

- $C = \mathcal{E}^T B^T D B \mathcal{E}$: CeedOperator

Composition of different operators defined on different element topologies possible

- $A = P^T C P$: User code responsible for parallelization on different compute devices. We use PETSc

Composition of solvers for multiphysics problems

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

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

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 $\mathbf{F}(\mathbf{u}) = \mathbf{0}$:

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

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

Ex: For a Dirichlet Stokes flow

$$\begin{aligned} \nabla \cdot \boldsymbol{\sigma} &= \mathbf{F}_b \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned} \Rightarrow \begin{pmatrix} J_{uu} & J_{pu}^T \\ J_{pu} & 0 \end{pmatrix}$$

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.

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", Errestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Errestrictudi, CEED_NOTRANSPOSE, basisx, weights);
CeedOperatorSetField(op_mass, "v", Errestrictu, 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", Errestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Errestrictudi, CEED_NOTRANSPOSE, basisx, weights);
CeedOperatorSetField(op_mass, "v", Errestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
```

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



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", Errestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Errestrictudi, CEED_NOTRANSPOSE, basisx, weights);
CeedOperatorSetField(op_mass, "v", Errestrictu, CEED_TRANSPOSE, basisu, CEED_VECTOR_ACTIVE);
```

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

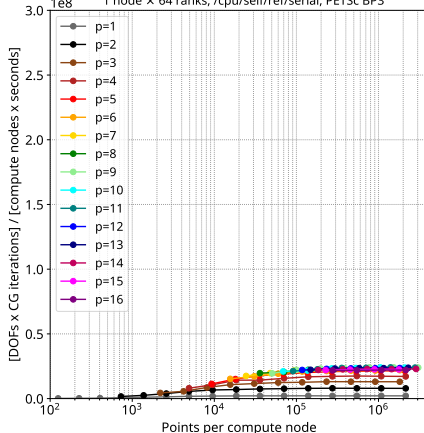
Composition of operators for multiphysics or mixed element meshes:

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



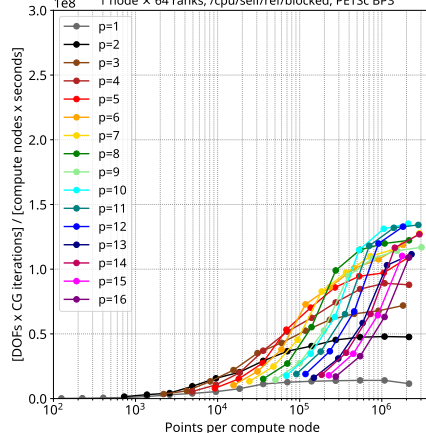
Performance w. r. t. space on KNL: ref

1e8 1 node × 64 ranks, /cpu/self/ref/serial, PETSc BP3



(a)

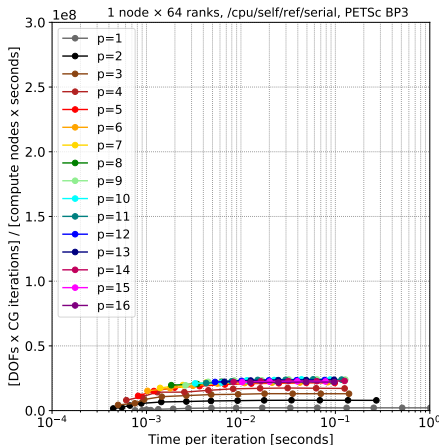
1e8 1 node × 64 ranks, /cpu/self/ref/blocked, PETSc BP3



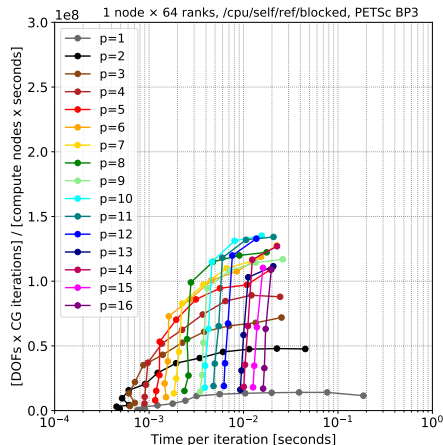
(b)

Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz)—Intel-18 compiler. In (a) serial implementation; in (b) blocked implementation ($q = P + 2$, $P = p + 1$)

Performance w. r. t. time on KNL: ref



(a)



(b)

Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz)—Intel-18 compiler. In (a) serial implementation; in (b) blocked implementation ($q = P + 2$, $P = p + 1$)



Performance w. r. t. space on Skylake: AVX

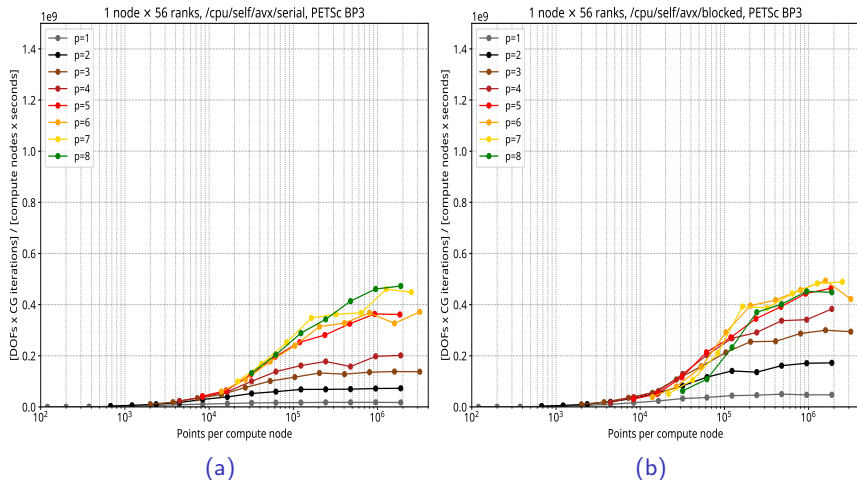


Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with intel-19 compiler.
Backends: in (a) AVX serial; in (b) AVX blocked ($q = P + 2$, $P = p + 1$)

Performance w. r. t. space, on Skylake: libXSMM

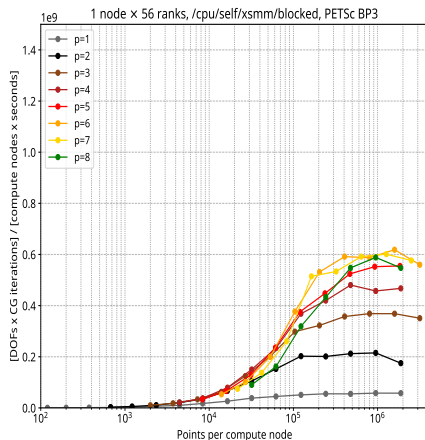
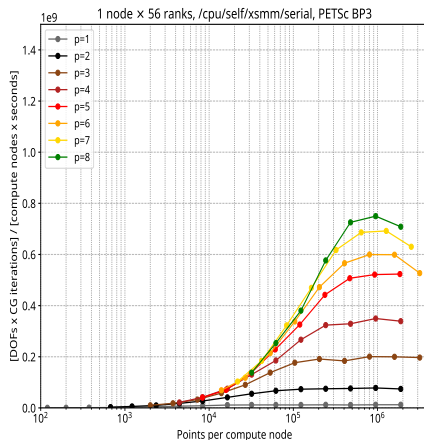
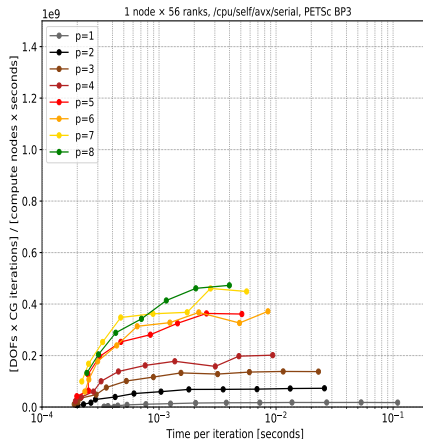


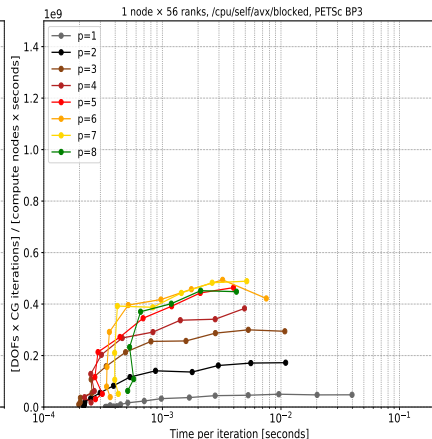
Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with intel-19 compiler.
Backends: in (a) libXSMM serial; in (b) libXSMM blocked ($q = P + 2$, $P = p + 1$)



Performance w. r. t. time on Skylake: AVX



(a)



(b)

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

Performance w. r. t. time, on Skylake: libXSMM

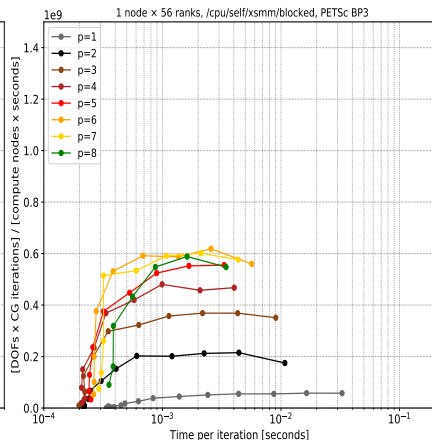
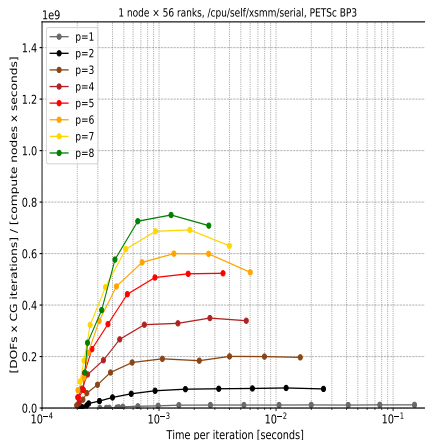
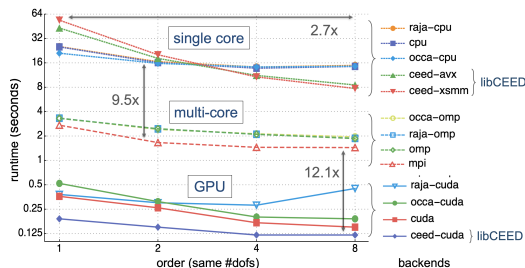


Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with intel-19 compiler.
Backends: in (a) libXSMM serial; in (b) libXSMM blocked ($q = P + 2$, $P = p + 1$)



Preliminary GPU results: MFEM + libCEED

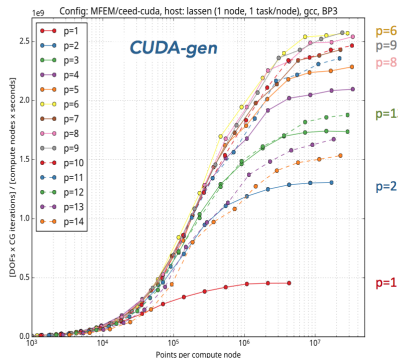
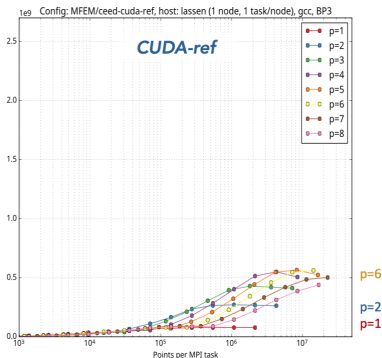


	1	2	4	8
occa-cuda =	00.52	00.31	00.20	00.19
raja-cuda =	00.38	00.30	00.28	00.45
cuda =	00.36	00.26	00.17	00.15
ceed-cuda =	<u>00.19</u>	<u>00.15</u>	<u>00.12</u>	<u>00.12</u>
occa-omp =	03.34	02.41	02.13	01.95
raja-omp =	03.32	02.45	02.10	01.87
omp =	03.30	02.46	02.10	01.86
mpi =	<u>02.72</u>	<u>01.66</u>	<u>01.45</u>	<u>01.44</u>
occa-cpu =	<u>21.05</u>	<u>15.77</u>	14.23	14.53
raja-cpu =	25.42	16.53	14.22	14.88
cpu =	25.18	16.11	13.73	14.45
ceed-avx =	43.04	18.16	11.20	08.53
ceed-xsmm =	53.80	20.13	<u>10.73</u>	<u>07.72</u>

single-GPU, multi-core CPU, and single-core CPU,
for 1.3 millions DOFs in 2D.

Results by Yohann Dudouit on a Linux desktop with a Quadro GV100 GPU,
sm_70, CUDA 10.1, and Intel Xeon Gold 6130 CPU @ 2.10GHz.

Preliminary GPU results: MFEM + libCEED (cont'ed)



Results by Yann Dudouit on Lassen (LLNL): CUDA-ref (left) and CUDA-gen (right) backends performance for 3D BP3 on a NVIDIA V100 GPU.

Towards a libCEED miniapp: a Navier-Stokes solver

Compressible Navier-Stokes equations in conservation form:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{u} = 0, \quad (1a)$$

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

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



Towards a libCEED miniapp: a Navier-Stokes solver

Compressible Navier-Stokes equations in conservation form:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{u} = 0, \quad (1a)$$

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

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

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:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{u} = 0, \quad (1a)$$

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

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

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

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

$$\mu \quad \leftarrow \text{dynamic viscosity}$$

$$g \quad \leftarrow \text{gravitational acceleration}$$

$$k \quad \leftarrow \text{thermal conductivity}$$

$$\lambda \quad \leftarrow \text{Stokes hypothesis constant}$$

$$c_p \quad \leftarrow \text{specific heat, constant pressure}$$

$$c_v \quad \leftarrow \text{specific heat, constant volume}$$



Vector form

The system (1) can be rewritten in vector form

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

for the state variables

$$\mathbf{q} = \begin{pmatrix} \rho \\ \mathbf{u} \equiv \rho \mathbf{u} \\ E \equiv \rho e \end{pmatrix} \begin{array}{l} \leftarrow \text{volume mass density} \\ \leftarrow \text{momentum density} \\ \leftarrow \text{energy density} \end{array} \quad (3)$$



Vector form

The system (1) can be rewritten in vector form

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

for the state variables

$$\mathbf{q} = \begin{pmatrix} \rho \\ \mathbf{u} \equiv \rho \mathbf{u} \\ E \equiv \rho e \end{pmatrix} \begin{array}{l} \leftarrow \text{volume mass density} \\ \leftarrow \text{momentum density} \\ \leftarrow \text{energy density} \end{array} \quad (3)$$

where

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

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

Space discretization

We use high-order finite/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/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 $\xi = (\xi, \eta, \zeta) \in \mathbf{I} = [-1, 1]^3$.

Define the discrete solution

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

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):

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

with $\mathcal{V}_p = \{v \in H^1(\Omega_e) \mid v \in P_p(\mathbf{I}), e = 1, \dots, N_e\}$.

Weak form:

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

Strong and weak formulations

The strong form of (3):

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

with $\mathcal{V}_p = \{v \in H^1(\Omega_e) \mid v \in P_p(\mathbf{I}), e = 1, \dots, N_e\}$.

Weak form:

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

For the Time Discretization we use an explicit formulation

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

solved with the adaptive Runge-Kutta-Fehlberg (RK4-5) method

A very simple example: The advection equation

We analyze the transport of total energy

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

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: 117649



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

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

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

where P_0 is the atmospheric pressure.

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

Density current

order: $p = 10$, $\Omega = [0, 6000]^2 \text{ m} \times [0, 3000] \text{ m}$, elem. resolution: 500 m,
Nodes: 893101



Application example prototype: 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

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

ρ ← density of air

$u_{1,\infty}$ ← unperturbed (far field) axial velocity

A_D ← area swept by rotor

C_T ← thrust coefficient



Application example prototype: 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

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

ρ $\leftarrow$ density of air

$u_{1,\infty}$ $\leftarrow$ unperturbed (far field) axial velocity

A_D $\leftarrow$ area swept by rotor

C_T $\leftarrow$ thrust coefficient

We project this nodal force onto the surrounding cells via

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



Application example prototype: 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

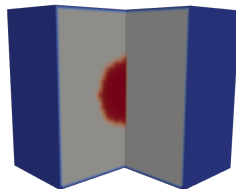
$$F_T = \frac{1}{2} \rho u_{1,\infty}^2 A_D C_T,$$

ρ $\leftarrow$ density of air

$u_{1,\infty}$ $\leftarrow$ unperturbed (far field) axial velocity

A_D $\leftarrow$ area swept by rotor

C_T $\leftarrow$ thrust coefficient



We project this nodal force onto the surrounding cells via

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

This adds a source/sink for momentum in the conservation equation

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = \mathbf{S}(\mathbf{q}), \quad \text{with } \mathbf{S}(\mathbf{q}) = (0, F_{\text{turbine}} \hat{\mathbf{i}} - \rho g \hat{\mathbf{k}}, 0).$$

Some work in progress

In collaboration with Ken Jensen (PHASTA) we are working on Streamline upwind/Petrov-Galerkin (SUPG) stabilization of our Navier-Stokes example:

Not stabilized version.

Stabilized version.

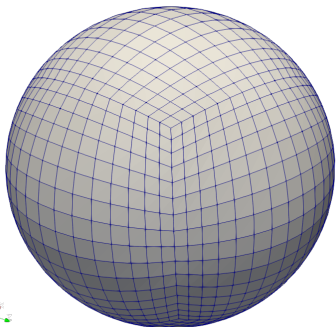


Some other work in progress

Converted BP1 (mass operator) & BP3 (Poisson's equation) on the cubed-sphere as a prototype for shallow-water equations solver

$$\frac{\partial \mathbf{u}}{\partial t} = -(\omega + f)\hat{\mathbf{k}} \times \mathbf{u} - \nabla \left(\frac{1}{2}|\mathbf{u}|^2 + g(h + h_s) \right) \quad (10a)$$

$$\frac{\partial h}{\partial t} = -\nabla \cdot (h_0 + h)\mathbf{u} \quad (10b)$$



Challenges:

Transform

$$\mathbf{x} = (x, y, z) \in \mathbb{R}^3 \hookrightarrow$$

$$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}) \in \mathbb{R}^2 \hookrightarrow$$

$$\xi = (\xi, \eta) \in \mathbf{I} = [-1, 1]^2$$



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
- We welcome contributors and users
<https://github.com/CEED/libCEED>



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

- We welcome contributors and users
<https://github.com/CEED/libCEED>

Ongoing and future work:

- Implicit-explicit (IMEX) and fully implicit time discretizations
 - NS solver & SWE solver
- BDDC and FDM preconditioners
- Algorithmic differentiation of Q-functions
- Ongoing work on CUDA and HIP optimizations.



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
- We welcome contributors and users
<https://github.com/CEED/libCEED>



Ongoing and future work:

- Implicit-explicit (IMEX) and fully implicit time discretizations
 - NS solver & SWE solver
- BDDC and FDM preconditioners
- Algorithmic differentiation of Q-functions
- Ongoing work on CUDA and HIP optimizations.

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

Thank you!