libCEED tutorial

Valeria Barra[1] and Jeremy Thompson[1,2]

[1] Department of Computer Science, CU Boulder
[2] Department of Applied Math, CU Boulder

(remotely for)
CEED 4 AM

August 12, 2020

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer?

CEED
EXASCALE DISCRETIZATIONS

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer? 

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer?
- A scientist that wants to solve a PDE fast?

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer? 
- A scientist that wants to solve a PDE fast? 

CEED
EXASCALE DISCRETIZATIONS

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer? 
- A scientist that wants to solve a PDE fast? 
- An algorithm developer that wants to evaluate/apply new FE methods?

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer? 
- A scientist that wants to solve a PDE fast? 
- An algorithm developer that wants to evaluate/apply new FE methods? 

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer? 
- A scientist that wants to solve a PDE fast? 
- An algorithm developer that wants to evaluate/apply new FE methods? 
- A solver developer that wants to easily compose FE tech?

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer?
- A scientist that wants to solve a PDE fast?
- An algorithm developer that wants to evaluate/apply new FE methods?
- A solver developer that wants to easily compose FE tech?

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer? 
- A scientist that wants to solve a PDE fast? 
- An algorithm developer that wants to evaluate/apply new FE methods? 
- A solver developer that wants to easily compose FE tech? 
- A perf engineer and/or vendor rep who wants to understand the essential algebraic needs of high-performance FE?

Is this tutorial for you?

Are you:

- A Finite Elements library or app developer?
- A scientist that wants to solve a PDE fast?
- An algorithm developer that wants to evaluate/apply new FE methods?
- A solver developer that wants to easily compose FE tech?
- A perf engineer and/or vendor rep who wants to understand the essential algebraic needs of high-performance FE?

Is this tutorial for you?

Are you:

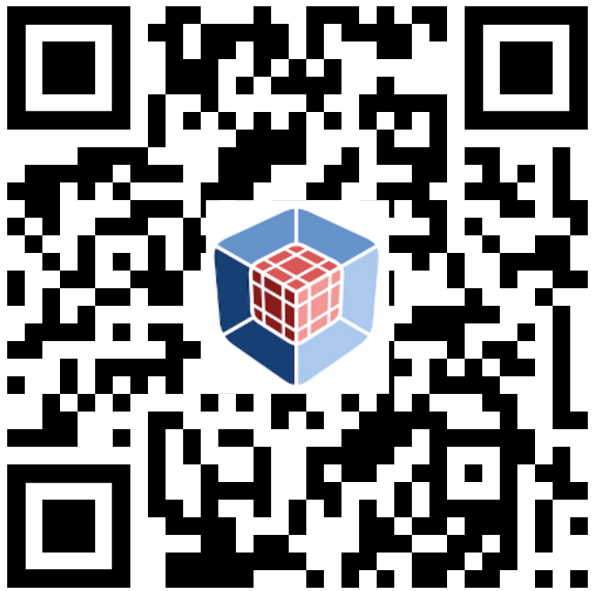
- A Finite Elements library or app developer? 😎
- A scientist that wants to solve a PDE fast? 😎
- An algorithm developer that wants to evaluate/apply new FE methods? 😎
- A solver developer that wants to easily compose FE tech? 😎
- A perf engineer and/or vendor rep who wants to understand the essential algebraic needs of high-performance FE? 😎

We've got you covered! 🚀

CEED
EXASCALE DISCRETIZATIONS

libCEED: the library within CEED (Center for Efficient Exascale Discretizations)

- Primary target: high-order finite/spectral element methods (FEM/SEM) exploiting tensor-product structure
- Open source (BSD-2 license) C library with Fortran and Python interfaces
- Releases: v0.1 (January 2018), v0.2 (March 2018), v.0.3 (September 2018), v0.4 (March 2019), v0.5 (September 2019), v0.6 (March 2020)

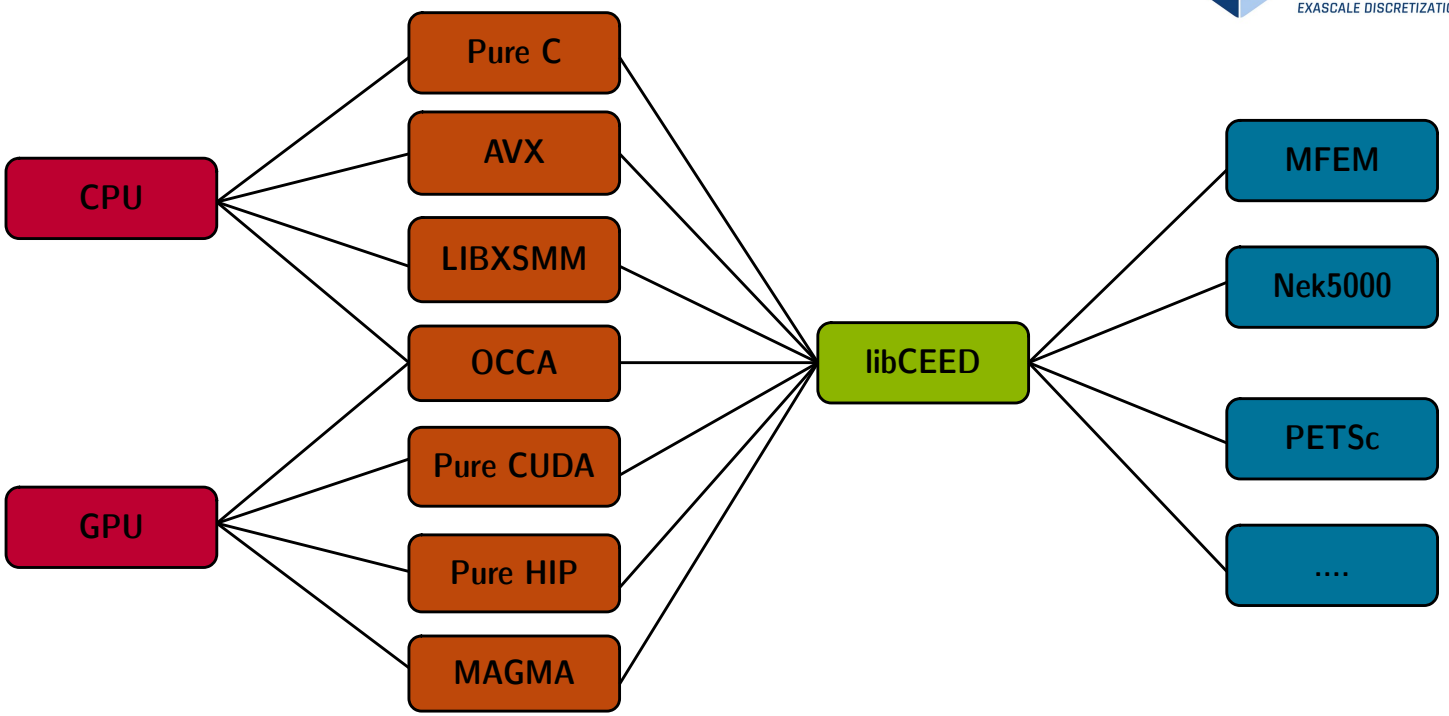


For latest release:

Kolev T., Fischer P., Abdelfattah A., Ananthan S., **Barra V.**, Beams N., Brown J. et al., *CEED ECP Milestone Report: Improve performance and capabilities of CEED-enabled ECP applications on Summit/Sierra* (2020, March 31st) DOI: `http://doi.org/10.5281/zenodo.3860804`

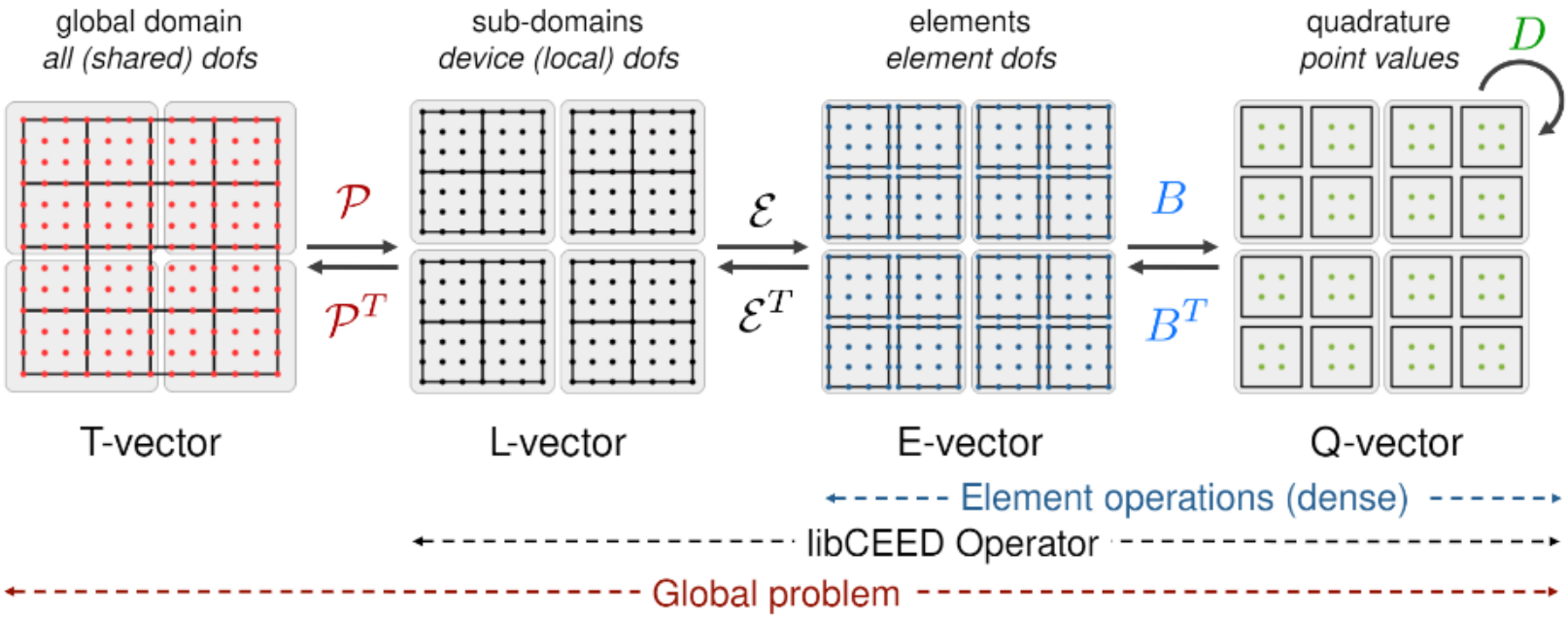
CEED EXASCALE DISCRETIZATIONS

libCEED backends



libCEED decomposition

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



- $\mathcal{E}$ - CeedElemRestriction, local gather/scatter
- B - CeedBasis, provides basis operations such as interp and grad
- D - CeedQFunction, representation of PDE at quadrature points
- A_L - CeedOperator, aggregation of libCEED objects

Point-wise QFunctions

User-defined QFunctions:

$$-\nabla \cdot (\kappa(\mathbf{x}) \nabla u)$$

Point-wise QFunctions

User-defined QFunctions:

$$-\nabla \cdot (\kappa(\mathbf{x}) \nabla u)$$

or from libCEED's Gallery:

$$\nabla \cdot (\nabla u)$$

are point-wise functions that do not depend on element resolution, topology, or basis order

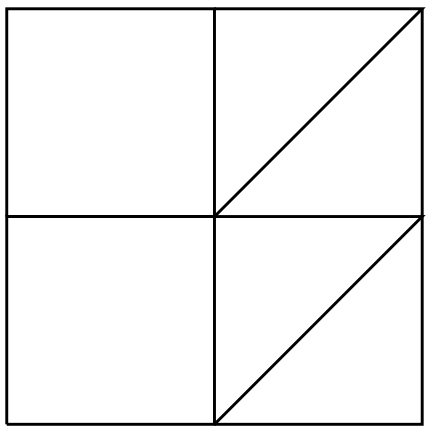
Point-wise QFunctions

User-defined QFunctions:

$$-\nabla \cdot (\kappa(\mathbf{x}) \nabla u)$$

or from libCEED's Gallery:

$$\nabla \cdot (\nabla u)$$



are point-wise functions that do not depend on element resolution, topology, or basis order

Point-wise QFunctions

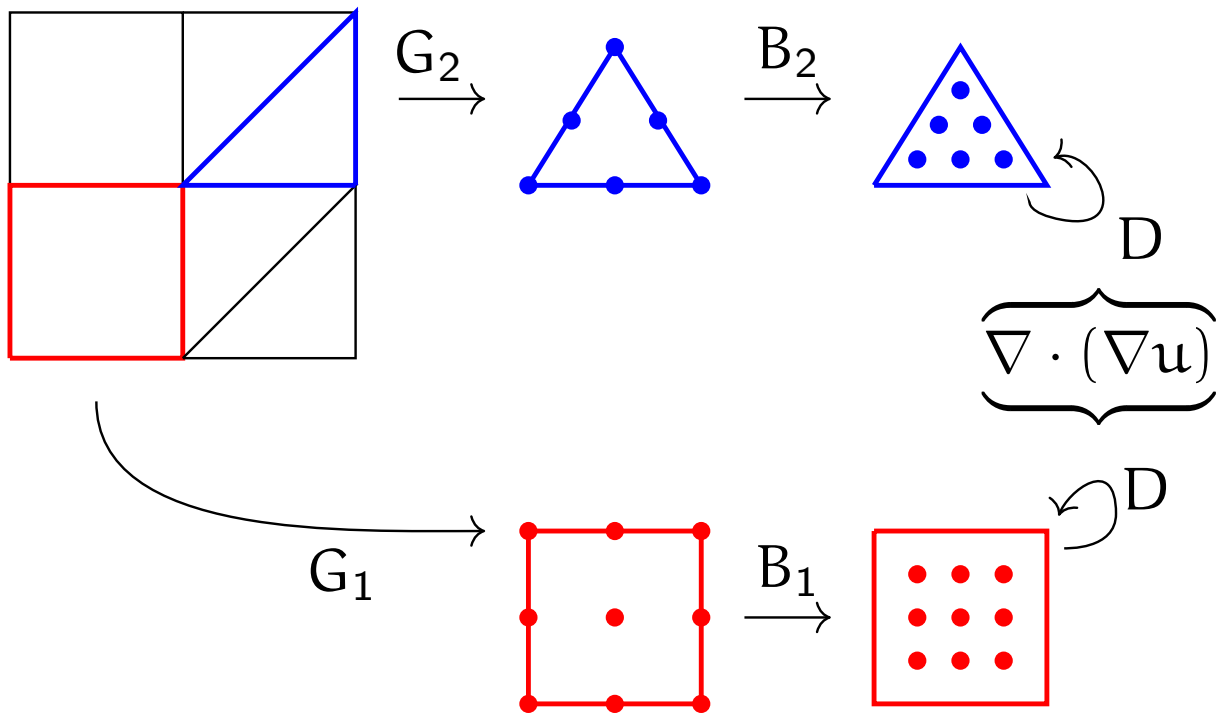
User-defined QFunctions:

$$-\nabla \cdot (\kappa(\mathbf{x}) \nabla u)$$

or from libCEED's Gallery:

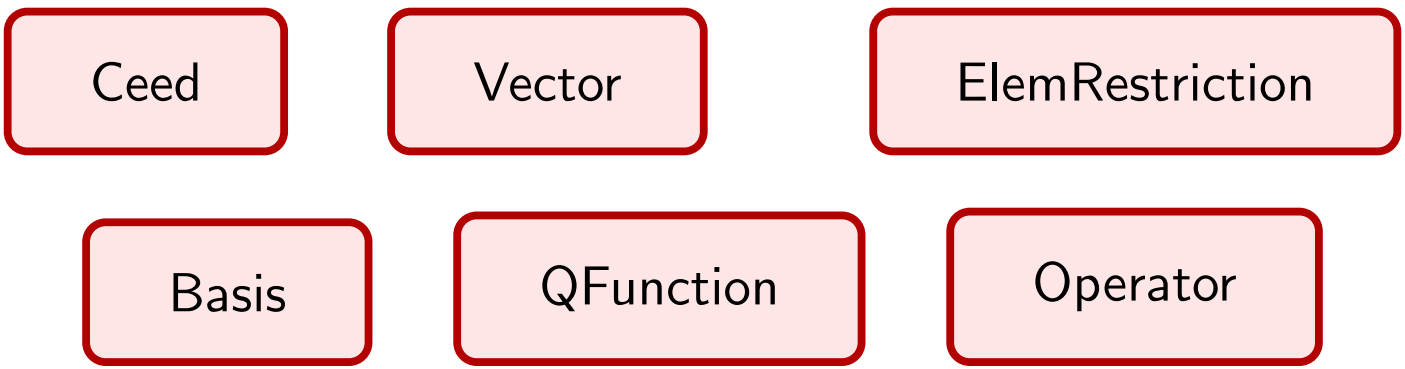
$$\nabla \cdot (\nabla u)$$

are point-wise functions that do not depend on element resolution, topology, or basis order

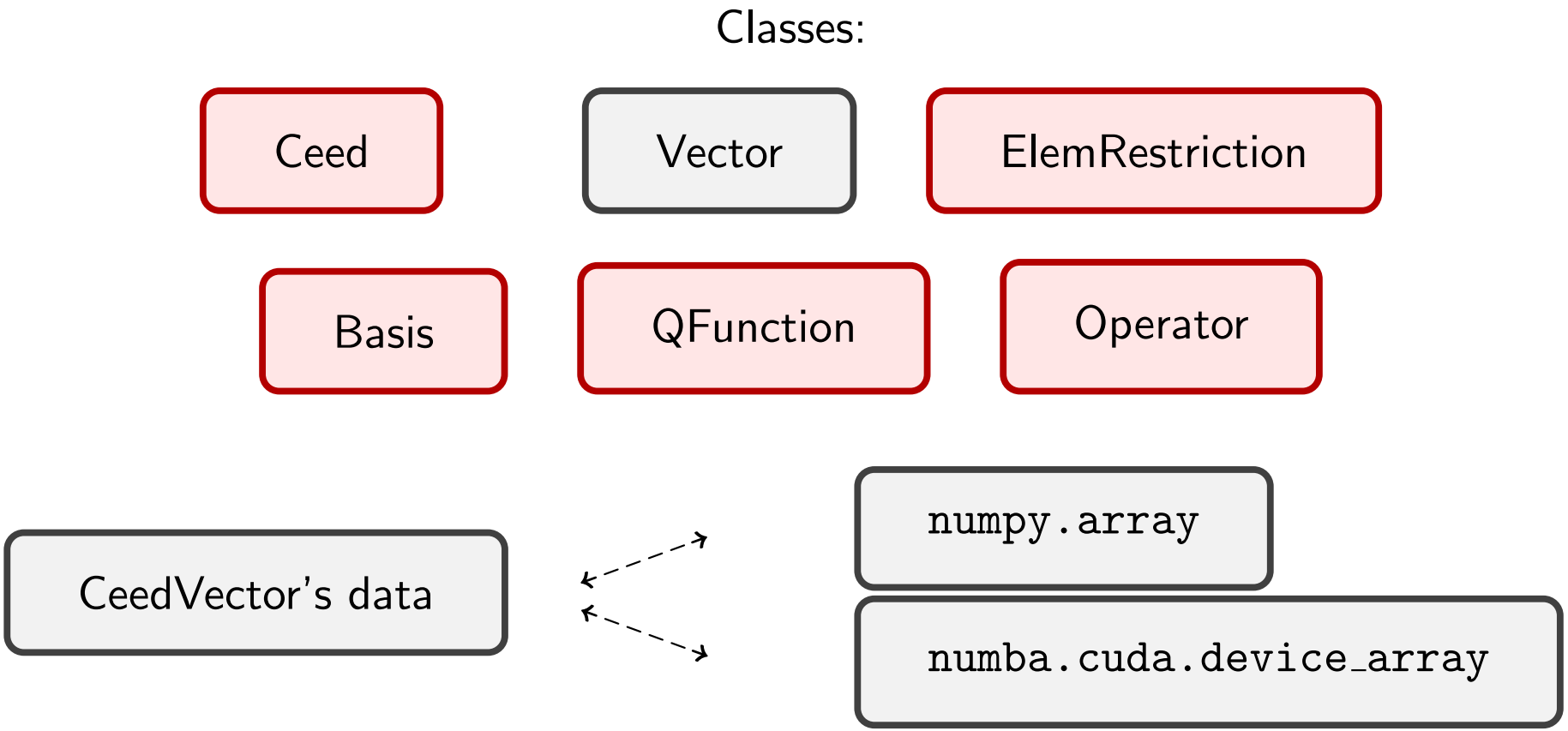


libCEED's Python interface

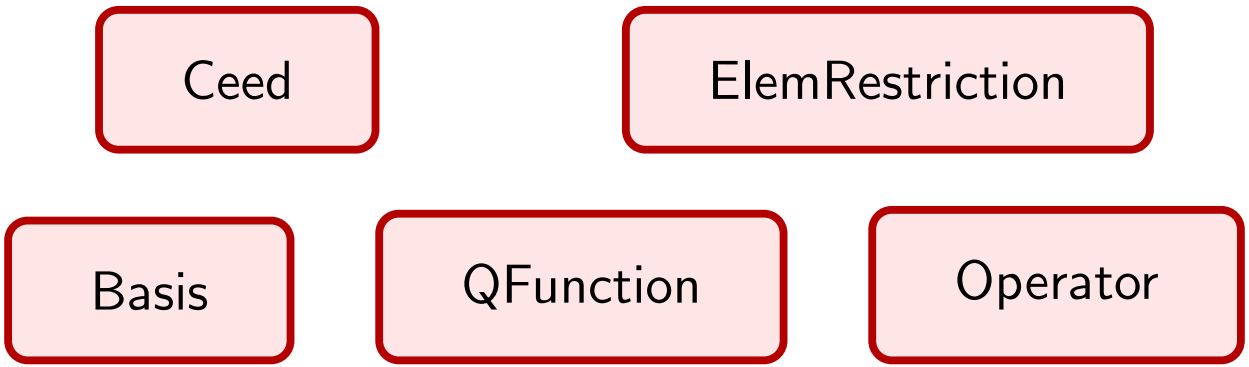
Classes:



libCEED's Python interface



libCEED's Python interface



More details:

Barra V., Brown J., Thompson J., Dudouit Y., *High-performance operator evaluations with ease of use: libCEED's Python interface*, Proceedings of the 19th Python in Science Conference (2020, July 12) DOI: `http://doi.org/10.25080/Majora-342d178e-00c`

Documentation

Our (very first!) user manual can be found at:

`https://libceed.readthedocs.io`

CEED EXASCALE DISCRETIZATIONS

libCEED's Python interface tutorials

More info on our Python interface and interactive Jupyter notebook tutorials can be found at:

`https://qrgo.page.link/YvwiP`

libCEED + PETSc

For the purpose of separation of concerns (e.g., mesh handling or time-stepping) we use PETSc for some hands-on exercises in this tutorial.

Note!

If you want to visualize the outputs produced by some of our miniapps in `.vtu` or `.vtk` format, it is recommended that you use Paraview or Visit.

If you need PETSc on your machine

PETSc can be found on ALCF, OLCF, NERSC facilities. But if you want to experiment locally on your machine:

```
$ git clone git@gitlab.com:petsc/petsc.git
$ cd petsc
$ export PETSC_DIR=$PWD

$ ./configure PETSC_ARCH=arch-tutorial-debug-no-fortran
--with-fortran-bindings=0

$ make PETSC_DIR=$PETSC_DIR \
   PETSC_ARCH=arch-tutorial-debug-no-fortran all

$ make PETSC_DIR=$PETSC_DIR \
   PETSC_ARCH=arch-tutorial-debug-no-fortran check
```

CEED
EXASCALE DISCRETIZATIONS

Download and build libCEED

Now we are ready to download and build libCEED in a sibling directory relative to the one were we have just built PETSc.

```
$ cd ../
$ git clone git@github.com:CEED/libCEED.git
$ cd libCEED
$ make
```

To build and run our libCEED+PETSc examples:

```
$ cd examples/petsc
$ make PETSC_DIR=$PETSC_DIR \
   PETSC_ARCH=arch-tutorial-debug-no-fortran
$./bpsraw
```

CEED
EXASCALE DISCRETIZATIONS

Online tutorial

If you can't/don't want to install locally on your machine, you can follow this tutorial online on Binder

```
https://bit.ly/
libceed-petsc-tutorial
```

Performance on an AMD EPYC: libXSMM

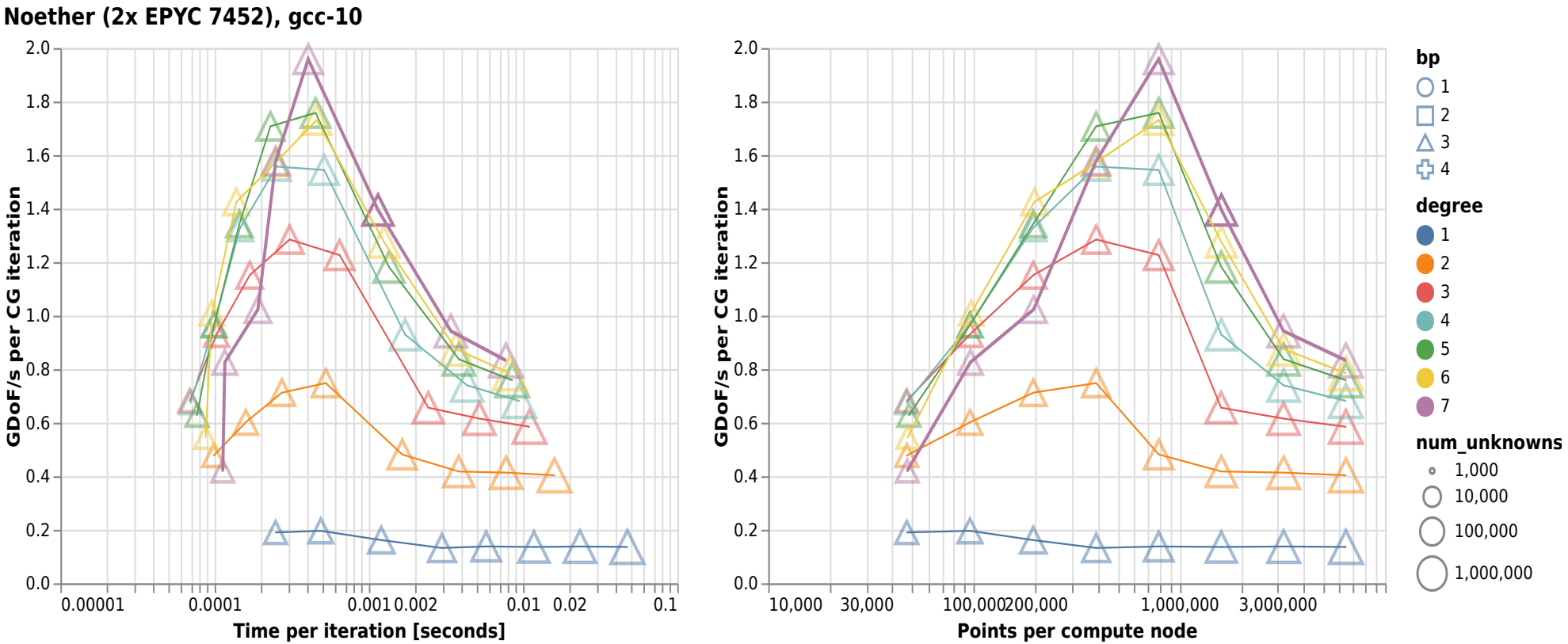
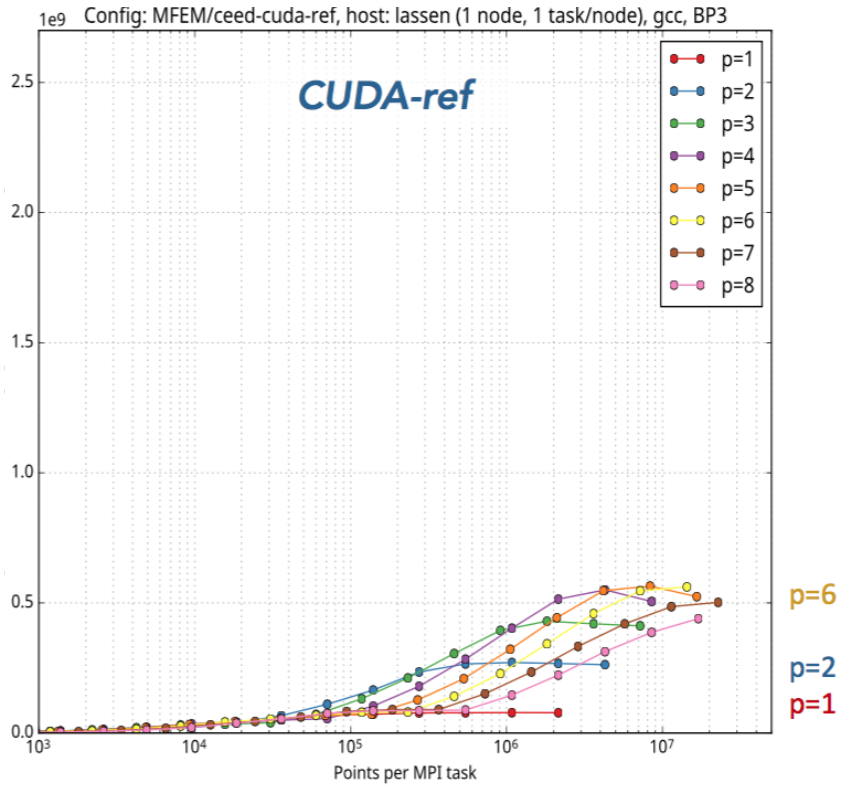
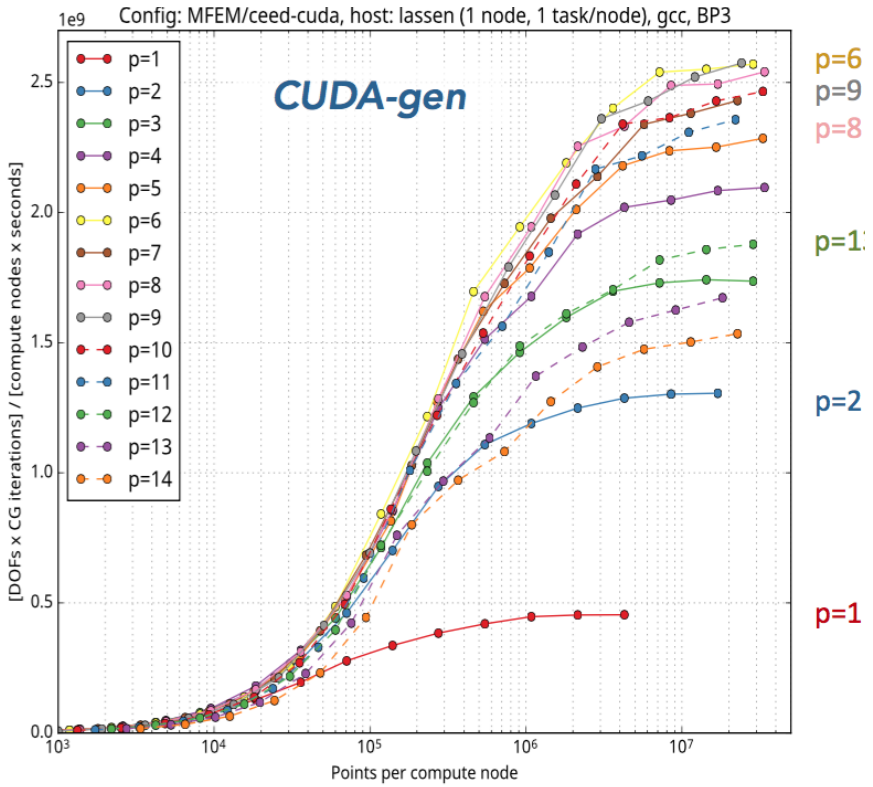


Figure: 2x AMD EPYC 7452 (32-core) with gcc-10 compiler. LIBXSMM blocked backend ($q = P + 1$, $P = p + 1$) with respect to time (left) and problem size (right)

GPU results: MFEM + libCEED





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

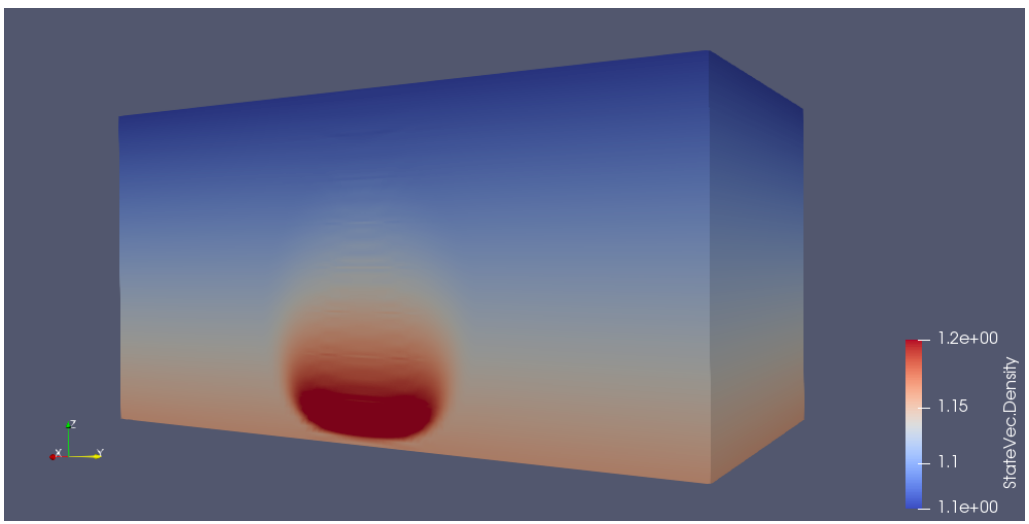
CFD miniapps: a compressible Navier-Stokes solver

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

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

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

where $\boldsymbol{\sigma} = \mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T + \lambda(\nabla \cdot \mathbf{u})\mathbf{I}_3)$, and
Eq. of state: $(c_p/c_v - 1)\,(E - \mathbf{U} \cdot \mathbf{U}/(2\rho) - \rho g z) = P$

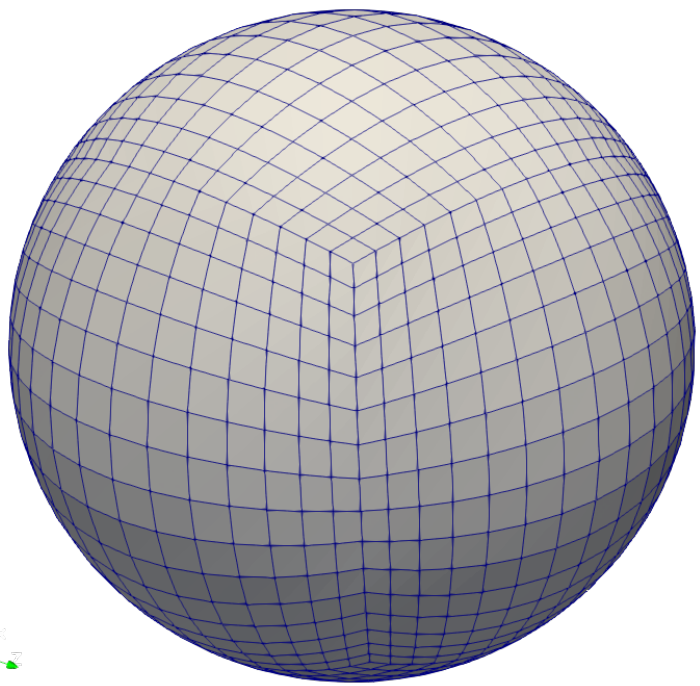


BPs on the cubed-sphere

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

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

$$\frac{\partial h}{\partial t} = -\nabla \cdot (h_0 + h)\mathbf{u} \tag{2b}$$

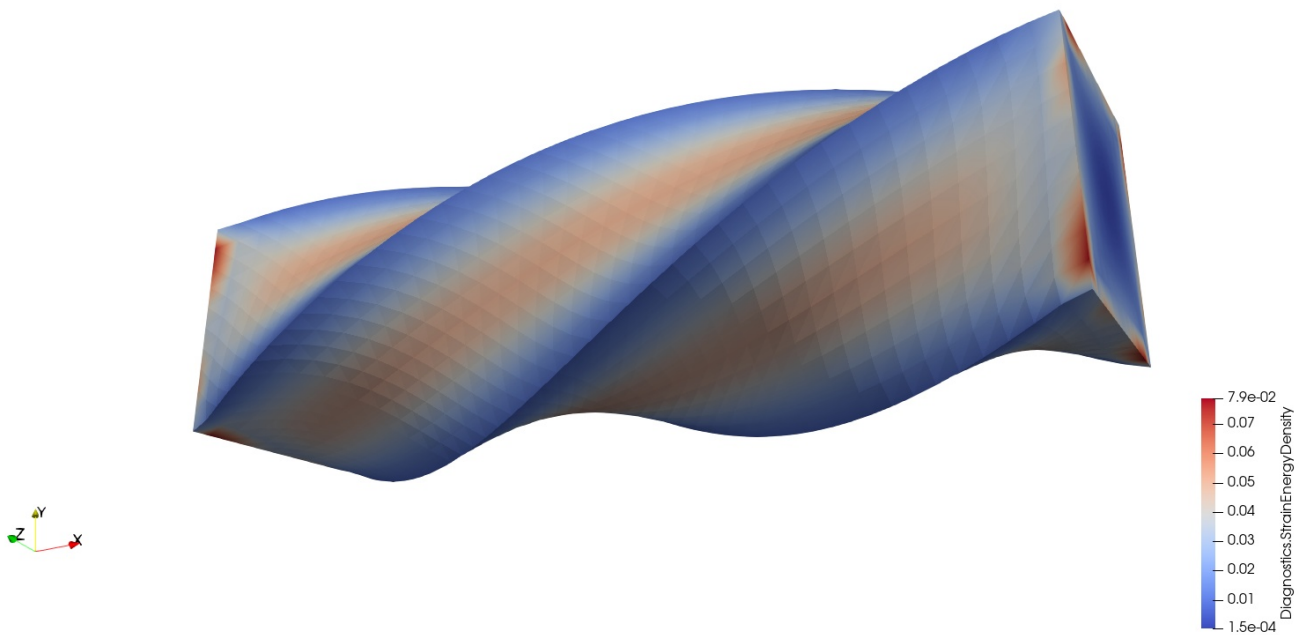


Solid mechanics miniapp

Arbitrary order solid mechanics mini-app on unstructured meshes

Three modes:

- Linear elasticity
- Neo-Hookean hyperelasticity at small strain
- Neo-Hookean hyperelasticity at finite strain



CEED EXASCALE DISCRETIZATIONS

Solid mechanics miniapp

Solver:

- PETSc SNES nonlinear solver with load increments
- Preconditioned CG on linearized problem

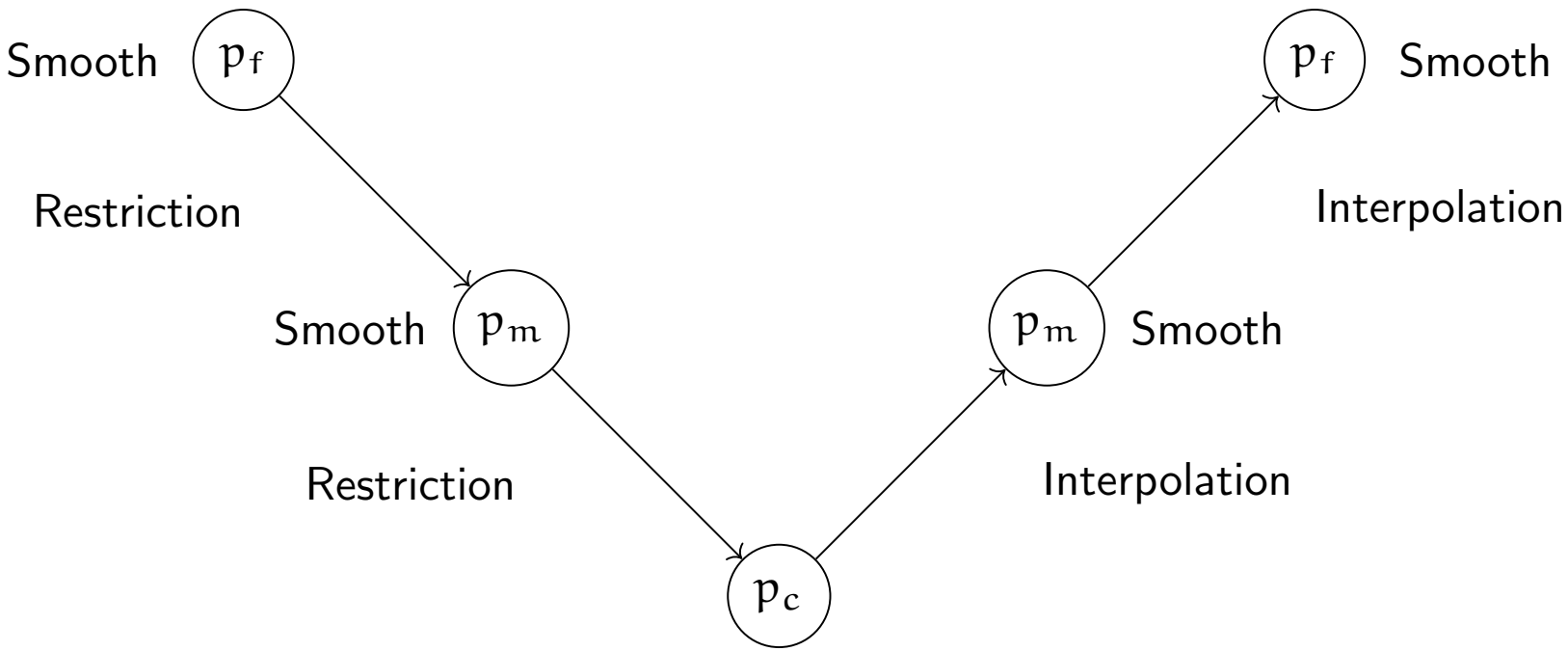
Preconditioning:

- p-multigrid with matrix-free transfer operators
- Jacobi smoothing with true operator diagonal
- AMG on the coarse level
- Runtime selection of CPU or GPU backends

CEED
EXASCALE DISCRETIZATIONS

p-multigrid

3 level multigrid with PETSc PCMG



Coarse Solve - AMG on liner elements

Smoother - Jacobi with operator diagonal

Grid transfer operators

Restriction / Interpolation is largely a basis operation

- General libCEED Operator
  $A_L = \mathcal{E}^T B^T D B \mathcal{E}$
- Interpolation Operator
  $A_{ctof} = \mathcal{E}_c^T I \frac{1}{m} B_{ctof} \mathcal{E}_f$
- Interpolation basis
  $B_{ctof} = R^{-1} Q^T B_c$
  $B_f = QR$

Current support

p-multigrid

```
CeedOperatorMultigridLevelCreate(CeedOperator opFine,
    CeedVector PMultFine,
    CeedElemRestriction rstrCoarse,
    CeedBasis basisCoarse, CeedOperator *opCoarse,
    CeedOperator *opProlong, CeedOperator *opRestrict)
```

Operator Diagonal

```
CeedOperatorLinearAssembleDiagonal(CeedOperator op,
    CeedVector assembled, CeedRequest *request)
```

Point Block Diagonal

```
CeedOperatorLinearAssemblePointBlockDiagonal(
    CeedOperator op, CeedVector assembled,
    CeedRequest *request)
```

CEED
EXASCALE DISCRETIZATIONS

Outlook

Ongoing and future work:

- Algorithmic differentiation of Q-Functions
- Ongoing work on CUDA and HIP optimizations
- Complete SWE solver on the cubed-sphere
- We always welcome contributors and users
  `https://github.com/CEED/libCEED`

CEED
EXASCALE DISCRETIZATIONS

Outlook

Ongoing and future work:

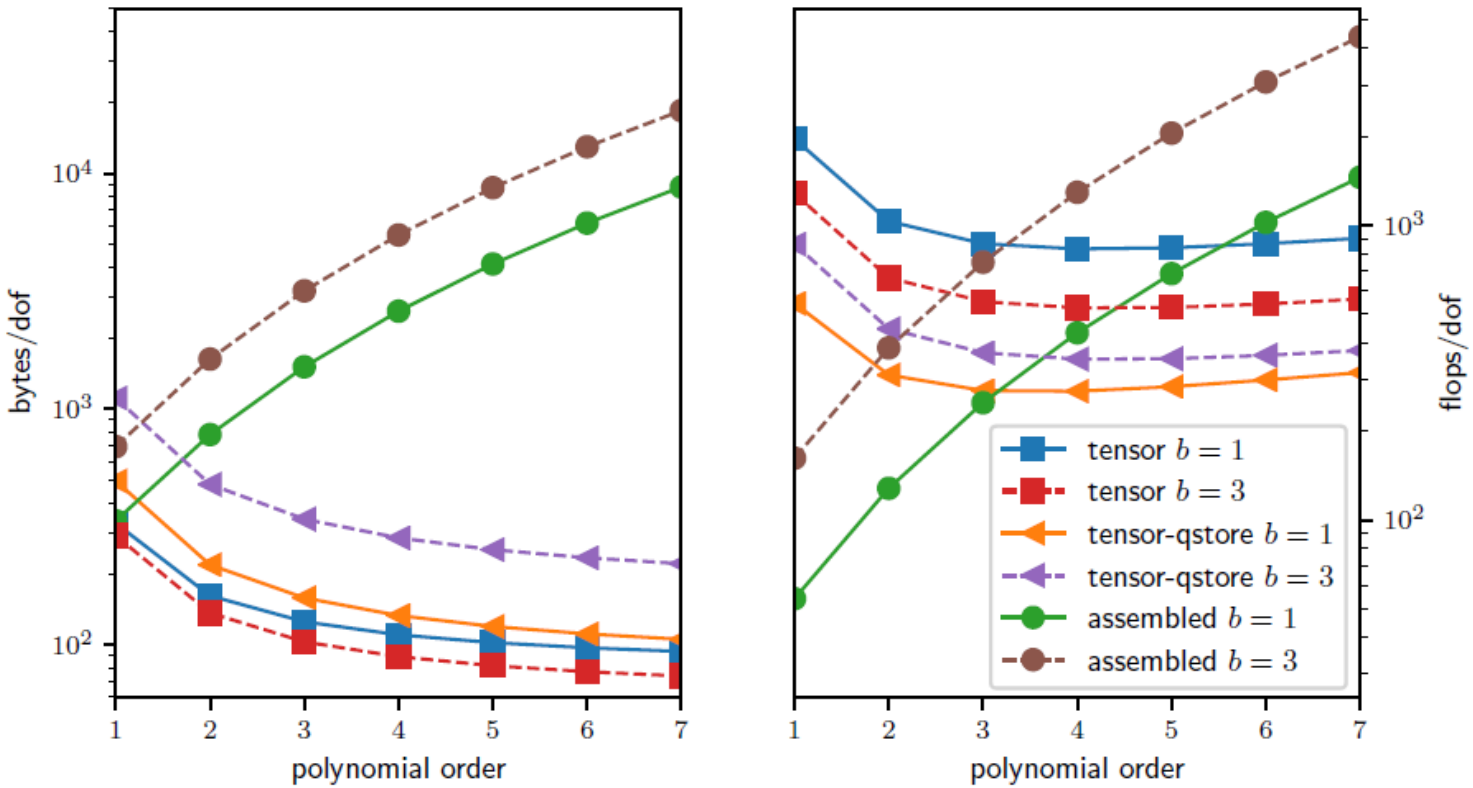
- Algorithmic differentiation of Q-Functions
- Ongoing work on CUDA and HIP optimizations
- Complete SWE solver on the cubed-sphere
- We always welcome contributors and users
  `https://github.com/CEED/libCEED`

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

Thank you!

CEED EXASCALE DISCRETIZATIONS

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]

CEED EXASCALE DISCRETIZATIONS

Vector form

The system (1) can be rewritten in vector form

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = S(\mathbf{q}) \,, \tag{3}$$

for the state variables

$$\mathbf{q} = \begin{pmatrix} \rho \\ \mathbf{U} \equiv \rho \mathbf{u} \\ E \equiv \rho e \end{pmatrix} \begin{matrix} \leftarrow \text{ volume mass density} \\ \leftarrow \text{ momentum density} \\ \leftarrow \text{ energy density} \end{matrix} \tag{4}$$

Vector form

The system (1) can be rewritten in vector form

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = S(\mathbf{q}) \,, \tag{3}$$

for the state variables

$$\mathbf{q} = \begin{pmatrix} \rho \\ \mathbf{U} \equiv \rho \mathbf{u} \\ E \equiv \rho e \end{pmatrix} \begin{matrix} \leftarrow \text{ volume mass density} \\ \leftarrow \text{ momentum density} \\ \leftarrow \text{ energy density} \end{matrix} \tag{4}$$

where

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

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

CEED
EXASCALE DISCRETIZATIONS

Space discretization

We use high-order finite/spectral elements: high-order Lagrange polynomials over non-uniformly spaced nodes, $\{x_i\}_{i=0}^{p}$, 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 $\mathbf{X} = (X, Y, Z) \in \mathbf{I} = [-1, 1]^3$.

CEED
EXASCALE DISCRETIZATIONS

Space discretization

We use high-order finite/spectral elements: high-order Lagrange polynomials over non-uniformly spaced nodes, $\{x_i\}_{i=0}^{p}$, 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 $\mathbf{X} = (X, Y, Z) \in \mathbf{I} = [-1, 1]^3$.

Define the discrete solution

$$\mathbf{q}_N(\mathbf{x}, t)^{(e)} = \sum_{k=1}^{P} \psi_k(\mathbf{x}) \mathbf{q}_k^{(e)} \tag{5}$$

with P the number of nodes in the element e.

We use tensor-product bases $\psi_{kji} = h_i(X) h_j(Y) h_k(Z)$.

CEED
EXASCALE DISCRETIZATIONS

Strong and weak formulations

The strong form of (4):

$$\int_\Omega v \left(\frac{\partial \mathbf{q}_N}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}_N) \right) \, d\Omega = \int_\Omega v \mathbf{S}(\mathbf{q}_N) \, d\Omega \,, \ \forall v \in \mathcal{V}_p \tag{6}$$

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

$$\int_\Omega v \frac{\partial \mathbf{q}_N}{\partial t} \, d\Omega + \int_\Gamma v \hat{\mathbf{n}} \cdot \mathbf{F}(\mathbf{q}_N) \, d\Omega - \int_\Omega \nabla v \cdot \mathbf{F}(\mathbf{q}_N) \, d\Omega = \int_\Omega v \mathbf{S}(\mathbf{q}_N) \, d\Omega \,, \ \forall v \in \mathcal{V}_p \tag{7}$$

CEED
EXASCALE DISCRETIZATIONS

Strong and weak formulations

The strong form of (4):

$$\int_\Omega v \left(\frac{\partial \mathbf{q}_N}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}_N) \right) \, d\Omega = \int_\Omega v \mathbf{S}(\mathbf{q}_N) \, d\Omega \,, \ \forall v \in \mathcal{V}_p \tag{6}$$

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

$$\int_\Omega v \frac{\partial \mathbf{q}_N}{\partial t} \, d\Omega + \int_\Gamma v \hat{\mathbf{n}} \cdot \mathbf{F}(\mathbf{q}_N) \, d\Omega - \int_\Omega \nabla v \cdot \mathbf{F}(\mathbf{q}_N) \, d\Omega = \int_\Omega v \mathbf{S}(\mathbf{q}_N) \, d\Omega \,, \ \forall v \in \mathcal{V}_p \tag{7}$$

Explicit time discretization:

$$\mathbf{q}_N^{n+1} = \mathbf{q}_N^n + \Delta t \sum_{i=1}^{s} b_i k_i \,, \tag{8}$$

adaptive Runge-Kutta-Fehlberg (RKF4-5) method

Implicit time discretization:

$$\begin{aligned} \mathbf{f}(\mathbf{q}_N) &\equiv \mathbf{g}(t^{n+1}, \mathbf{q}_N, \dot{\mathbf{q}}_N) = 0 \,, \\ \dot{\mathbf{q}}_N(\mathbf{q}_N) &= \alpha \mathbf{q}_N + \mathbf{z}_N \end{aligned} \tag{9}$$

α-method

CEED EXASCALE DISCRETIZATIONS

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

$$\rho = \frac{P_0}{(c_p - c_v)\theta(\mathbf{x}, t)} \pi(\mathbf{x}, t)^{\frac{c_v}{c_p - c_v}}, \tag{10a}$$

$$e = c_v \theta(\mathbf{x}, t)\pi(\mathbf{x}, t) + \mathbf{u} \cdot \mathbf{u}/2 + gz, \tag{10b}$$

where P_0 is the atmospheric pressure.

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

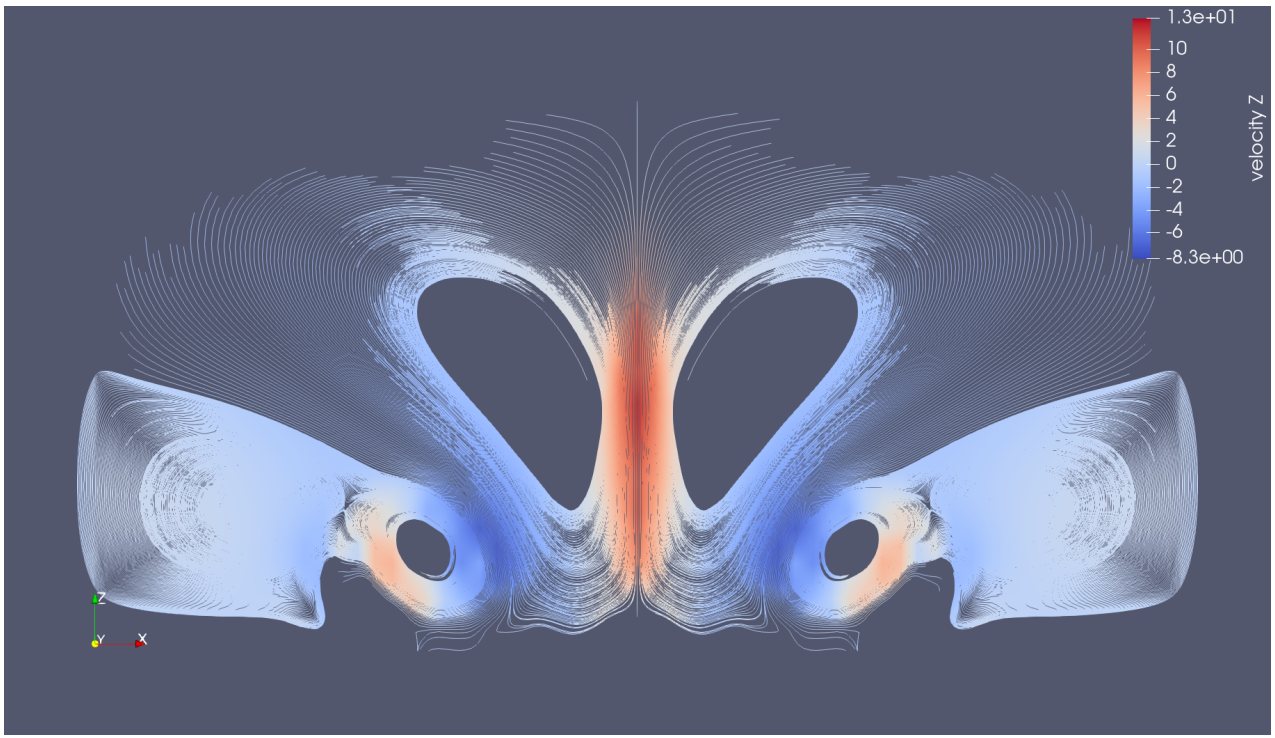
CEED
EXASCALE DISCRETIZATIONS

Density current

order: $p = 10$, $\Omega = [0, 6000]^2$ m $\times [0, 3000]$ m, elem. resolution: 500 m, FEM nodes: 893101

CEED
EXASCALE DISCRETIZATIONS

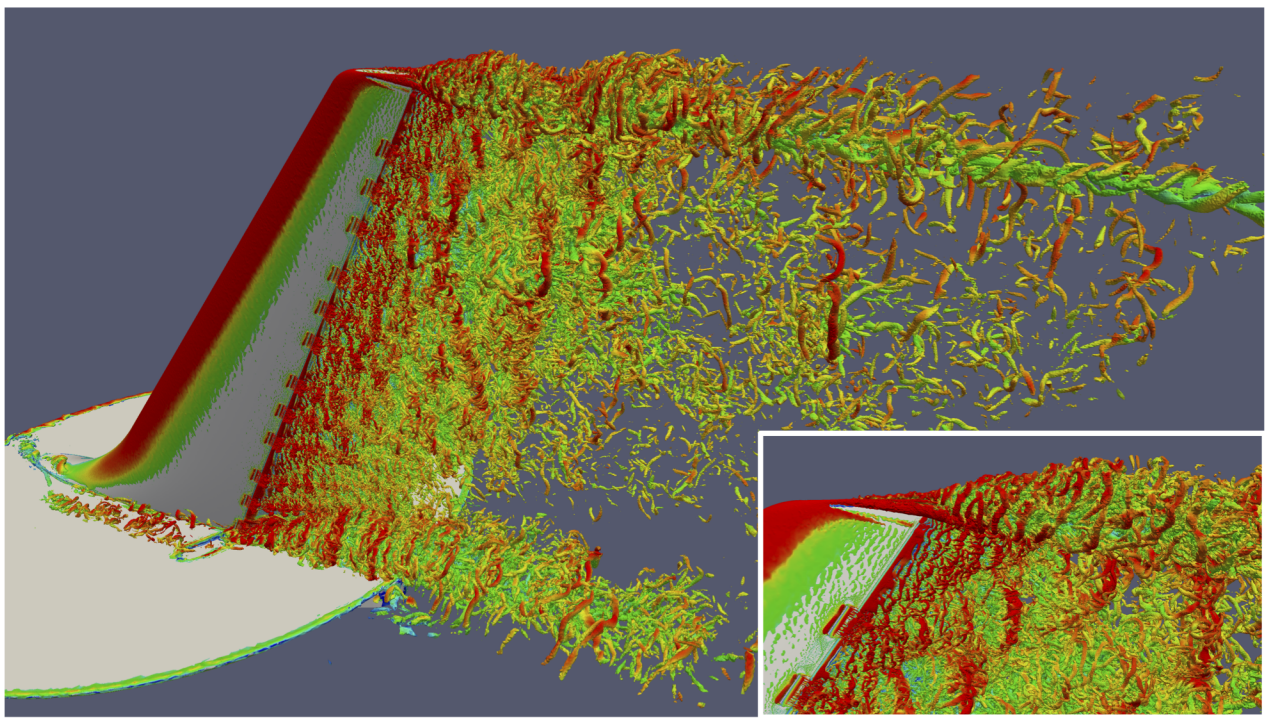
Recent Developments: Implicit time-stepping



CEED
EXASCALE DISCRETIZATIONS

Recent Developments: PHASTA Integration

In collaboration with PHASTA (FastMath) we have worked on libCEED's integration.



[Ref: `phasta.scigap.org`]

CEED EXASCALE DISCRETIZATIONS

Recent Developments: Stabilization methods

We have added Streamline Upwind (SU) and Streamline Upwind/Petrov-Galerkin (SUPG) stabilization methods to our Navier-Stokes example.

For the advection case:

Not stabilized version.

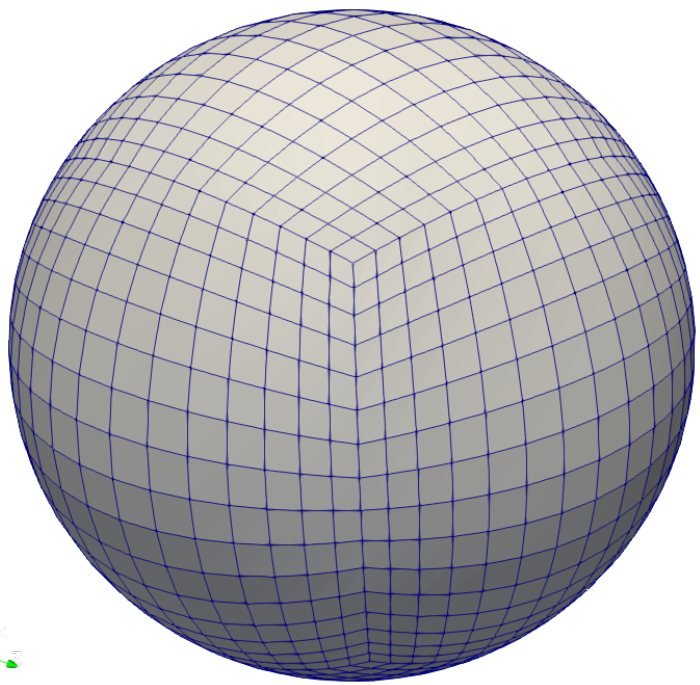
Stabilized version.

Recent Developments: BPs on the cubed-sphere

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

$$\frac{\partial \mathbf{u}}{\partial t} = -(\omega + f)\hat{k} \times \mathbf{u} - \nabla\left(\frac{1}{2}|\mathbf{u}|^2 + g(h + h_s)\right) \tag{11a}$$

$$\frac{\partial h}{\partial t} = -\nabla \cdot (h_0 + h)\mathbf{u} \tag{11b}$$



Conclusions

- We have showed libCEED's performance portability on several architectures, when integrated with PETSc and MFEM
- We have demonstrated the use of libCEED with PETSc for the numerical high-order solutions of
  - Full compressible Navier-Stokes equations
- We have included implicit time-stepping and SU/SUPG stabilization methods

CEED
EXASCALE DISCRETIZATIONS

libCEED backends

`/cpu/self/ref/*`:	with `*` reference serial and blocked implementations
`/cpu/self/avx/*`:	AVX (Advanced Vector Extensions instruction sets) with `*` reference serial and blocked implementations
`/cpu/self/xsmm/*`:	LIBXSMM (Intel library for small dense/sparse mat-multiply) with `*` reference serial and blocked implementations
`/*/occa`:	OCCA (just-in-time compilation) with `*`: CPU, GPU, OpenMP (Open Multi-Processing: API), OpenCL (framework for CPUs, GPUs, etc.)
`/gpu/magma`:	CUDA MAGMA (dense Linear Algebra library for GPUs and multicore architectures) kernels
`/gpu/cuda/*`:	CUDA with `*`: `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

CEED
EXASCALE DISCRETIZATIONS

Tensor contractions

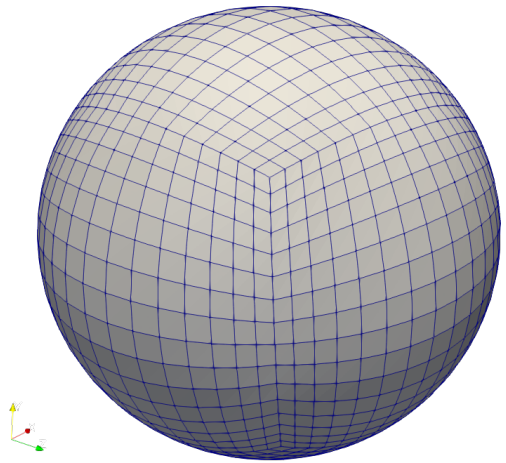
Let $\{x_i\}_{i=0}^{p}$ denote the LGL nodes with the corresponding interpolants $\{\psi_i^p\}_{i=0}^{p}$. Choose a quadrature rule with nodes $\{q_i^Q\}_{i=0}^{Q}$ and weights $\{w_i^Q\}$. The basis evaluation, derivative, and integration matrices are $B_{ij}^{Qp} = \psi_j^p(q_i^Q)$, $D_{ij}^{Qp} = \partial_x \psi_j^p(q_i^Q)$, and $W_{ij}^{Q} = w_i^Q \delta_{ij}$. In 3D:

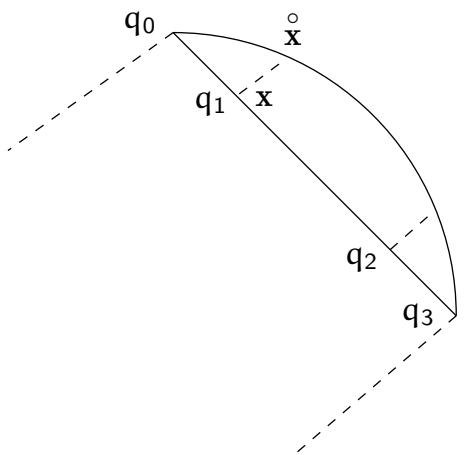
$$\mathbf{B} = B \otimes B \otimes B \tag{12}$$
$$\mathbf{D}_0 = D \otimes B \otimes B \tag{13}$$
$$\mathbf{D}_1 = B \otimes D \otimes B \tag{14}$$
$$\mathbf{D}_2 = B \otimes B \otimes D \tag{15}$$
$$\mathbf{W} = W \otimes W \otimes W \tag{16}$$

These tensor-product operations cost $2(p^3Q + p^2Q^2 + pQ^3)$ and touch only $O(p^3 + Q^3)$ memory. In the spectral element method, when the same LGL points are reused for quadrature (i.e., a collocated method with $Q = p + 1$), then $\mathbf{B} = \mathbf{I}$ and $\mathbf{D}$ reduces to $O(p^4)$.

CEED
EXASCALE DISCRETIZATIONS

Geometry on the sphere





Transform $\mathring{\mathbf{x}} = (\mathring{x}, \mathring{y}, \mathring{z})$ on the sphere $\hookrightarrow$
$\mathbf{x} = (x, y, z)$ on the discrete surface $\hookrightarrow$
$\mathbf{X} = (X, Y) \in \mathbf{I} = [-1, 1]^2$

$$\underset{(3\times 2)}{\frac{\partial \mathring{\mathbf{x}}}{\partial \mathbf{X}}} = \underset{(3\times 3)}{\frac{\partial \mathring{\mathbf{x}}}{\partial \mathbf{x}}} \underset{(3\times 2)}{\frac{\partial \mathbf{x}}{\partial \mathbf{X}}}$$

$$|J| = \left| \mathrm{col}_1\left(\frac{\partial \mathring{\mathbf{x}}}{\partial \mathbf{X}}\right) \times \mathrm{col}_2\left(\frac{\partial \mathring{\mathbf{x}}}{\partial \mathbf{X}}\right) \right|$$

CEED EXASCALE DISCRETIZATIONS