Composition of efficient, scalable solvers for performance-portable applications

Valeria Barra[1]

Collaborators: Jed Brown[1], Jeremy Thompson[1,2], Yohann Dudouit[3]

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

(remotely for)
Computational Research Division, Lawrence Berkeley National Laboratory

March 16, 2020

CEED EXASCALE DISCRETIZATIONS

ECP EXASCALE COMPUTING PROJECT

Where do I come from?

Siena, Tuscany, Italy

Ok, where do I come from, mathematically speaking?

Master's thesis:
Catmull-Clark *Subdivision Surfaces* (Computer-Aided Design, ′78)

The limit-surface obtained is a bicubic uniform B-spline surface of class C^2, except at *extraordinary* points

University of Colorado Boulder

PhD dissertation: Viscoelastic fluids

PhD in Applied Math from NJIT on numerical simulations of thin films (long-waves) of viscoelastic fluids




Viscoelastic materials:

- hysteresis: loop in stress-strain rate curve
- stress relaxation: constant $\epsilon \Rightarrow$ decreasing σ
- creep: constant $\sigma \Rightarrow$ increasing ϵ

University of Colorado Boulder

PhD dissertation

Second order finite differences on a staggered grid

Fixed Δx, adaptive Δt, Crank-Nicolson scheme

University of Colorado Boulder

PhD dissertation (cont'ed)

Long-wave approximation in two spatial dimensions:

$$(1+\lambda_2\partial_t)h_t + \frac{\partial}{\partial x}\left\{(\lambda_2-\lambda_1)\left(\frac{h^2}{2}Q - hR\right)h_t + \left[(1+\lambda_1\partial_t)\frac{h^3}{3} + (1+\lambda_2\partial_t)bh^2\right]\frac{\partial}{\partial x}\left(\frac{\partial^2 h}{\partial x^2}+\Pi(h)\right)\right\} = 0\,,$$

$$Q + \lambda_2 Q_t = -\frac{\partial}{\partial x}\left(\frac{\partial^2 h}{\partial x^2}+\Pi(h)\right),$$
$$R + \lambda_2 R_t = -h\frac{\partial}{\partial x}\left(\frac{\partial^2 h}{\partial x^2}+\Pi(h)\right).$$

disjoining pressure: $\Pi(h) = \frac{\gamma(1-\cos\theta_e)}{Mh_\star}\left[\left(\frac{h_\star}{h}\right)^n - \left(\frac{h_\star}{h}\right)^m\right]$,
θ_e contact angle, $M = 0.5$, $(n = 3, m = 2)$, $h_\star$ precursor film thickness.

Jeffreys' constitutive law:

$$\boldsymbol{\sigma} + \lambda_1\partial_t\boldsymbol{\sigma} = 2\eta(\dot{\boldsymbol{\epsilon}} + \lambda_2\partial_t\dot{\boldsymbol{\epsilon}})$$

University of Colorado Boulder

PhD dissertation (cont'ed)

Shear and extensional free-boundary flows of viscoelastic membranes

Linear finite elements with plane stress formulation. Different constitutive models considered: elastic, viscous, viscoelastic (Maxwell)

University of Colorado Boulder

References

Some references:

V. Barra, S. Afkhami, L. Kondic, *Mathematical and numerical modeling of thin viscoelastic films of Jeffreys type subjected to the van der Waals and gravitational forces*, EPJE, **42**, 1 – 14 (2019)

V. Barra, S. A. Chester, S. Afkhami, *Numerical Simulations of Nearly Incompressible Viscoelastic Membranes*, Computers & Fluids, **175**, 36 – 47 (2018)

V. Barra, S. Afkhami, L. Kondic, *Interfacial dynamics of thin viscoelastic films and drops*, J. Non-Newt. Fluid Mech. **237**, 26 – 38 (2016)

Internship at Pixar

12-week internship program at PIXAR

Developed 2 proprietary libraries in C++ for viscous fluid simulations on curved surfaces

A vorticity-formulation Navier-Stokes solver with fluid-structure interactions, using Discrete Exterior Calculus (DEC) and time-splitting

University of Colorado Boulder

Internship at Pixar

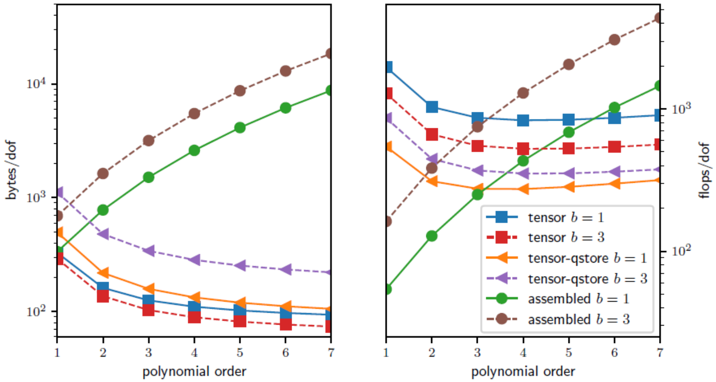
A thin film (long-wave) solver on curved surfaces

Arbitrary topology
and element shapes

Prototyped a plugin
for Side FX Houdini

Overview

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]

University of Colorado Boulder

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 $\mathrm{O}(\mathrm{p}^6)$ (for sparse matrix) to $\mathrm{O}(\mathrm{p}^4)$, and memory storage from $\mathrm{O}(\mathrm{p}^6)$ to $\mathrm{O}(\mathrm{p}^3)$
- We demonstrate the usage of libCEED, its integration with other packages, and some PETSc application examples

University of Colorado Boulder

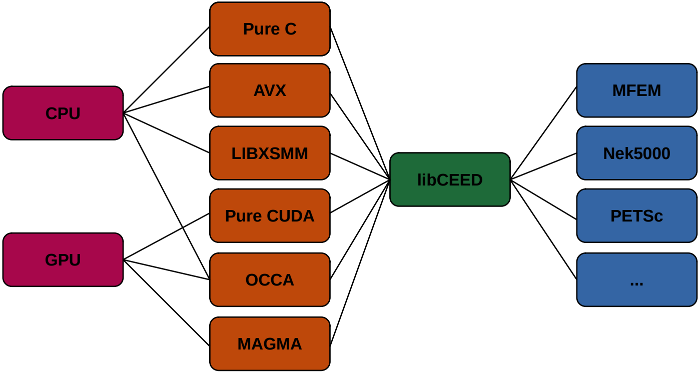
libCEED: the library of CEED (Center for Efficient Exascale Discretizations)

- 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)

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 2^{nd}) DOI: `https://doi.org/10.5281/zenodo.3477618`

libCEED backends



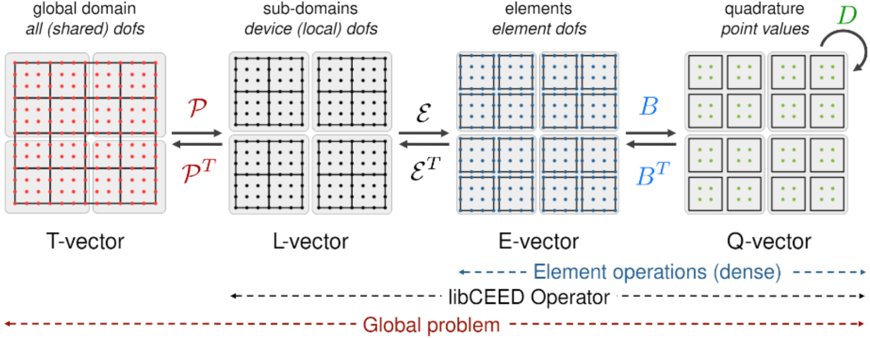
libCEED backends

/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

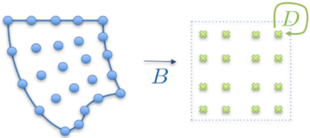
Same source code can call multiple CEEDs with different backends. On-device operator implementation with unique interface

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

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

University of Colorado Boulder

Point-wise QFunctions

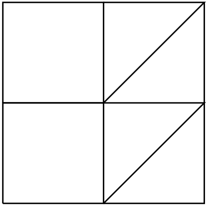
User-defined QFunctions:

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

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

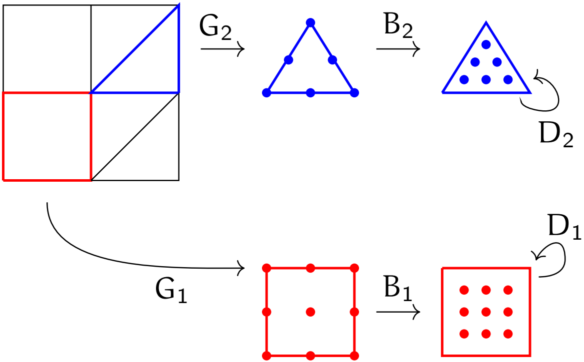
User-defined QFunctions:

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

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 API for operator composition

Creation of QFunctions and CEED operators:

```
CeedQFunctionCreateInterior(ceed, 1, Mass, Mass_loc, &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, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Erestrictudi, CEED_BASIS_COLLOCATED, qdata);
CeedOperatorSetField(op_mass, "v", Erestrictu, basisu, CEED_VECTOR_ACTIVE);
```

libCEED API for operator composition

Creation of QFunctions and CEED operators:

```
CeedQFunctionCreateInterior(ceed, 1, Mass, Mass_loc, &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, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Erestrictudi, CEED_BASIS_COLLOCATED, qdata);
CeedOperatorSetField(op_mass, "v", Erestrictu, basisu, CEED_VECTOR_ACTIVE);
```

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

University of Colorado Boulder

libCEED API for operator composition

Creation of QFunctions and CEED operators:

```
CeedQFunctionCreateInterior(ceed, 1, Mass, Mass_loc, &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, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Erestrictudi, CEED_BASIS_COLLOCATED, qdata);
CeedOperatorSetField(op_mass, "v", Erestrictu, basisu, CEED_VECTOR_ACTIVE);
```

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

Composition of operators for multiphysics or mixed element meshes:

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

University of Colorado Boulder

libCEED's Python interface

Classes:

Ceed

Vector

ElemRestriction

Basis

QFunction

Operator

libCEED's Python interface

Classes:

Ceed

Vector

ElemRestriction

Basis

QFunction

Operator

CeedVector's data ←----→ `numpy.array`

libCEED's Python interface

Classes:

Ceed | Vector | ElemRestriction

Basis | QFunction | Operator

CeedVector's data ←- - - -→ `numpy.array`

```
import libceed
ceed = libceed.Ceed()
n = 10
x = ceed.Vector(n)

print(x)
```

or

```
import libceed
ceed = libceed.Ceed()
x = ceed.Vector(size=10)

print(x)
```

University of Colorado Boulder

Python interface (some examples)

Vector

```
import libceed
ceed = libceed.Ceed('/gpu/cuda/gen')
x = ceed.Vector(size=10)

a = np.arange(1, 4, dtype="float64")
x.set_array(a, cmode=libceed.USE_POINTER)
b = x.get_array_read()
print(b)
```

```
import libceed
ceed = libceed.Ceed('/cpu/self')
u = ceed.Vector(size=10)

u.set_value(0)
print(u)
```

University of Colorado Boulder

Python interface (some examples)

Vector

```
import libceed
ceed = libceed.Ceed('/gpu/cuda/gen')
x = ceed.Vector(size=10)

a = np.arange(1, 4, dtype="float64")
x.set_array(a, cmode=libceed.USE_POINTER)
b = x.get_array_read()
print(b)
```

```
import libceed
ceed = libceed.Ceed('/cpu/self')
u = ceed.Vector(size=10)

u.set_value(0)
print(u)
```

Elemrestriction

```
ne = 8
x = ceed.Vector(ne+1)
y = ceed.Vector(2*ne)
r = ceed.ElemRestriction(ne, 2, ne+1, 1, ind, cmode=libceed.USE_POINTER)
r.apply(x, y)
r.T.apply(y, x)
```

C interface Vs Python interface

```
CeedQFunctionCreateInterior(ceed, 1, Mass, Mass_loc, &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, basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_mass, "weights", Erestrictudi, CEED_BASIS_COLLOCATED, qdata);
CeedOperatorSetField(op_mass, "v", Erestrictu, basisu, CEED_VECTOR_ACTIVE);
```

```
qf_mass = ceed.QFunction(1, qfs.apply_mass, os.path.join(file_dir,
"test-qfunctions.h:apply_mass"))
qf_mass.add_input("u", 5, libceed.EVAL_INTERP)
qf_mass.add_input("weights", 1, libceed.EVAL_NONE)
qf_mass.add_output("v", 5, libceed.EVAL_INTERP)

op_mass = ceed.Operator(qf_mass)
op_mass.set_field("u", Erestrictu, basisu, libceed.VECTOR_ACTIVE)
op_mass.set_field("weights", Erestrictudi, libceed.BASIS_COLLOCATED, qdata)
op_mass.set_field("v", Erestrictu, basisu, libceed.VECTOR_ACTIVE)
```

University of Colorado Boulder

Performance w. r. t. problem size on KNL: ref

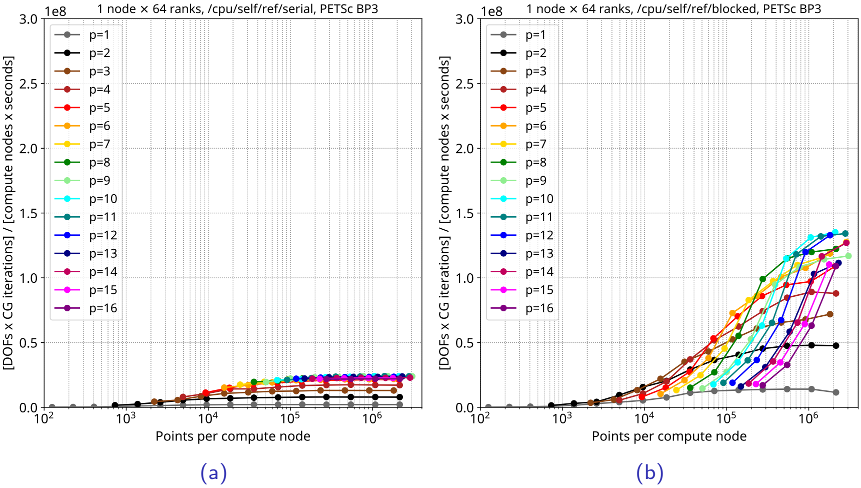


Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz) with Intel-18 compiler. In (a) serial implementation; in (b) blocked implementation ($q = P + 2$, $P = p + 1$)

University of Colorado Boulder

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

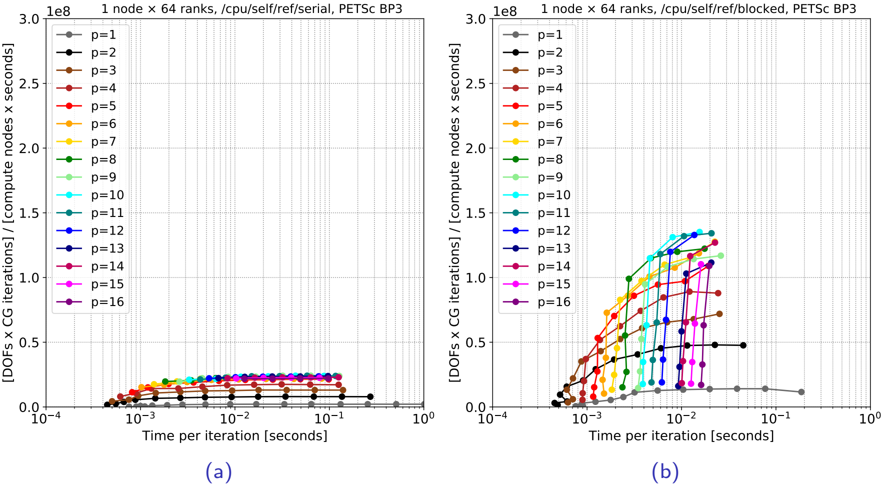


Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz) with Intel-18 compiler. In (a) serial implementation; in (b) blocked implementation ($q = P + 2$, $P = p + 1$)

Performance w. r. t. problem size on Skylake: AVX

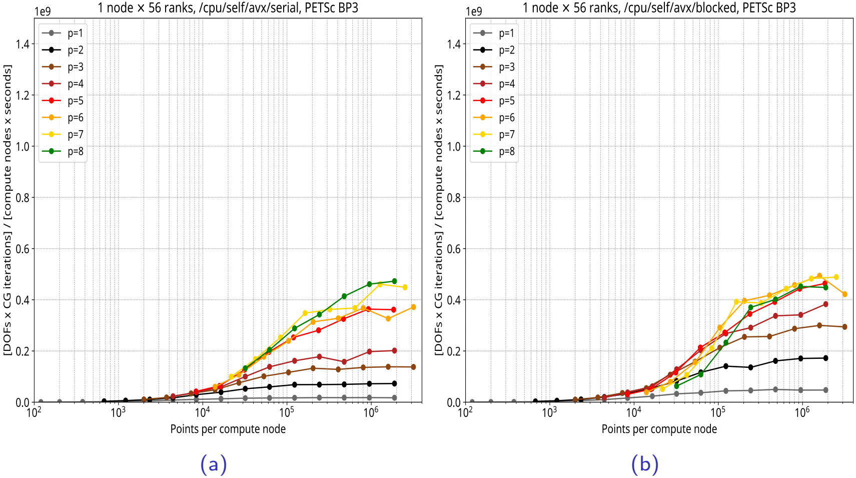


Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with gcc-8 compiler. Backends: in (a) AVX serial; in (b) AVX blocked ($q = P + 2$, $P = p + 1$)

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

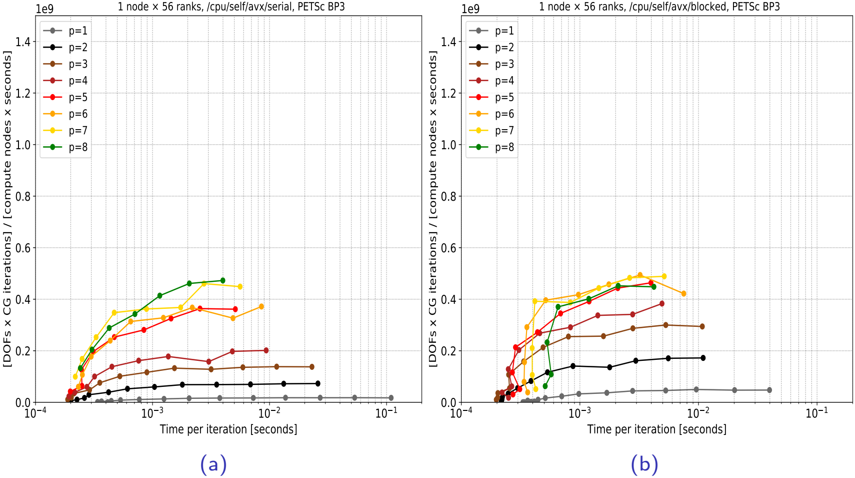


Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with gcc-8 compiler. Backends: in (a) AVX serial; in (b) AVX blocked ($q = P + 2$, $P = p + 1$)

Performance w. r. t. problem size on Skylake: libXSMM

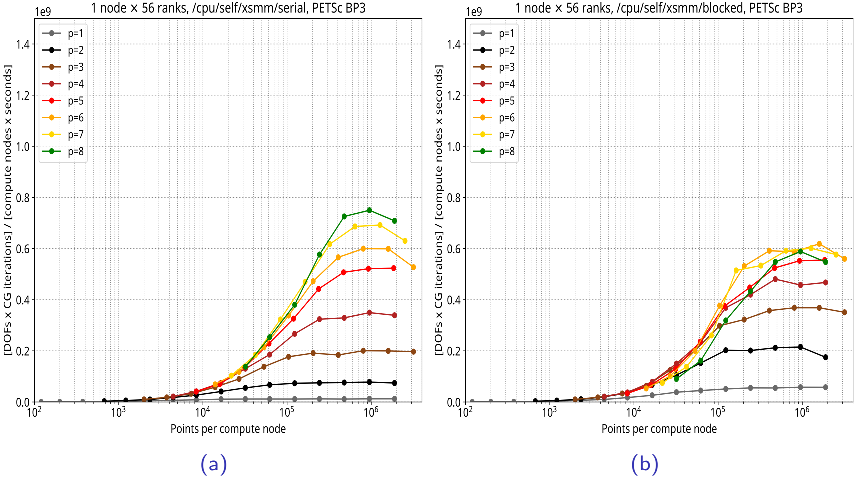


Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with gcc-8 compiler. Backends: in (a) libXSMM serial; in (b) libXSMM blocked ($q = P + 2$, $P = p + 1$)

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

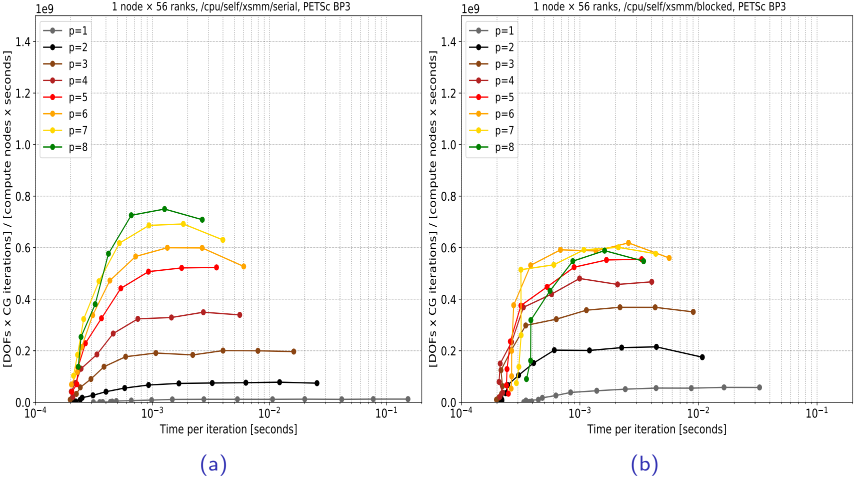


Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz) with gcc-8 compiler. Backends: in (a) $\mathrm{libXSMM}$ serial; in (b) $\mathrm{libXSMM}$ blocked ($q = P + 2$, $P = p + 1$)

Performance w. r. t. size on Summit: ref (non optimized)

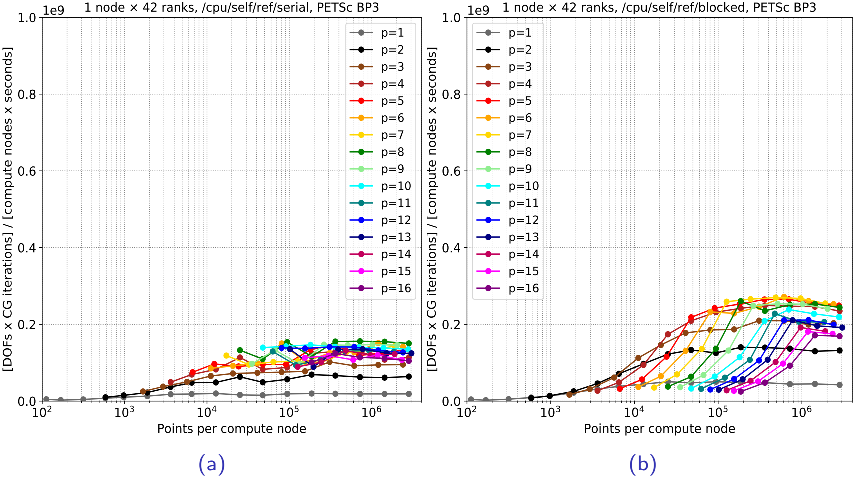
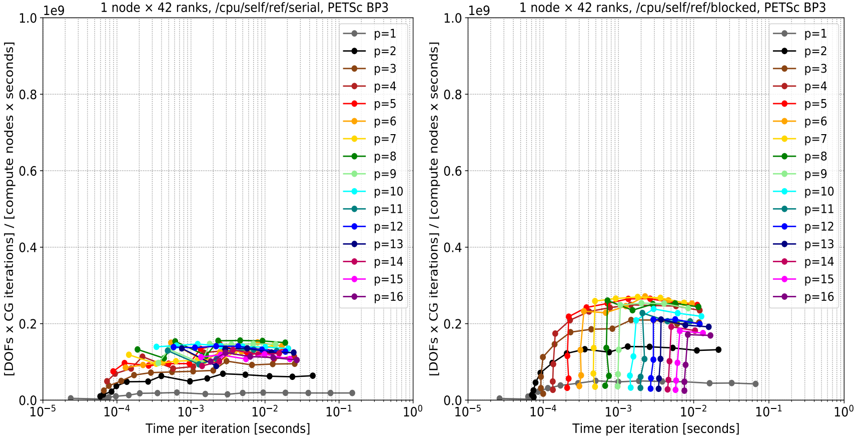


Figure: OLCF Summit (2x IBM POWER9) with gcc-9 compiler. Backends: in (a) ref serial; in (b) ref blocked ($q = P + 2$, $P = p + 1$)

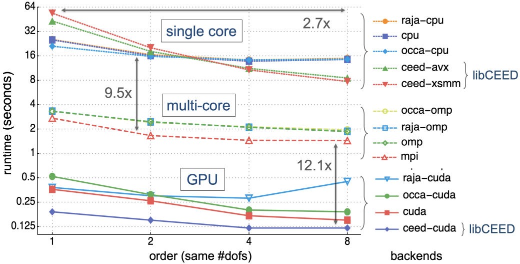
Performance w. r. t. time on Summit: ref (non optimized)



(a) (b)

Figure: OLCF Summit (2x IBM POWER9) with gcc-9 compiler. Backends: in (a) ref serial; in (b) ref blocked ($q = P + 2$, $P = p + 1$)

Preliminary GPU results: MFEM + libCEED



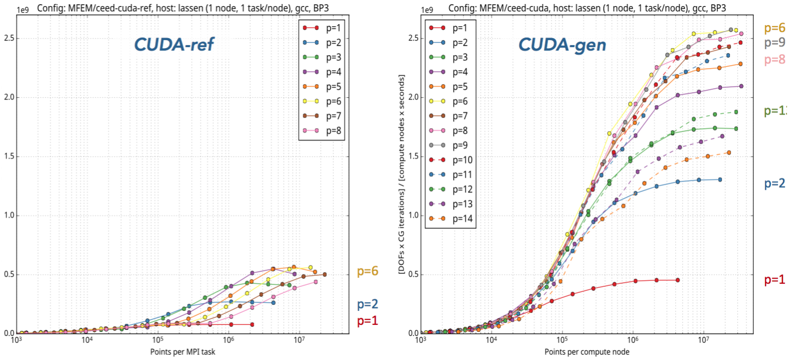
p	1	2	4	8
occa-cuda =	00.52	00.31	00.20	00.19
raja-cuda =	00.38	00.30	00.28	00.45
cuda =	00.36	00.26	00.17	00.15
ceed-cuda =	00.19	00.15	00.12	00.12
occa-omp =	03.34	02.41	02.13	01.95
raja-omp =	03.32	02.45	02.10	01.87
omp =	03.30	02.46	02.10	01.86
mpi =	02.72	01.66	01.45	01.44
occa-cpu =	21.05	15.77	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	10.73	07.72

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, $\mathrm{sm\_70}$, CUDA 10.1, and Intel Xeon Gold 6130 CPU @ 2.10GHz.

University of Colorado Boulder

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



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

University of Colorado Boulder

Towards a libCEED miniapp: a Navier-Stokes solver

Compressible Navier-Stokes equations in conservation form:

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

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

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

University of Colorado Boulder

Towards a libCEED miniapp: a Navier-Stokes solver

Compressible Navier-Stokes equations in conservation form:

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

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

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

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

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

- $\mu \leftarrow$ 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

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

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

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

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

where

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

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

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 $\mathbf{X} = (X, Y, Z) \in \mathbf{I} = [-1, 1]^3$.

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

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

with P the number of nodes in the element e.

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

University of Colorado Boulder

Strong and weak formulations

The strong form of (3):

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

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

Weak form:

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

Strong and weak formulations

The strong form of (3):

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

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

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

For the Time Discretization we started with an explicit formulation (recently expanded to implicit)

$$\mathbf{q}_N^{n+1} = \mathbf{q}_N^n + \Delta t \sum_{i=1}^{s} b_i k_i\,, \tag{7}$$

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

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

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

University of Colorado Boulder

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, $\pi(\boldsymbol{x}, t)$, and potential temperature, $\theta(\boldsymbol{x}, t)$, that relate to the state variables via

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

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

where P_0 is the atmospheric pressure.

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

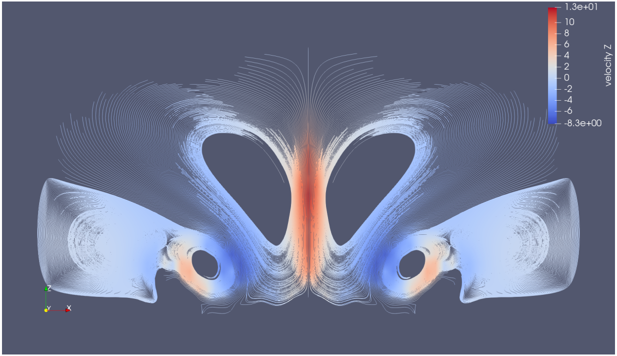
University of Colorado Boulder

Density current

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

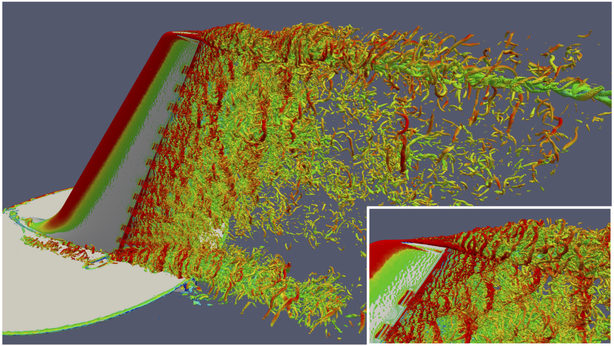
University of Colorado Boulder

Recent Developments: Implicit time-stepping



Recent Developments: PHASTA Integration

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



[Ref: phasta.scigap.org]

University of Colorado Boulder

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.

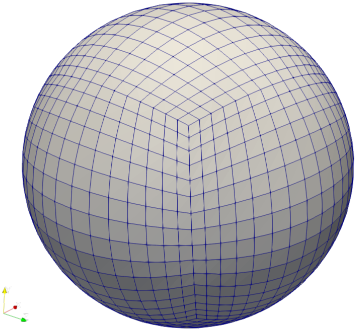
University of Colorado Boulder

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

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

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



University of Colorado Boulder

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
  - Advection equation
  - Full compressible Navier-Stokes equations
- We have included implicit time-stepping and SU/SUPG stabilization methods

University of Colorado Boulder

Outlook

Ongoing and future work:

- Our (very first!) user manual
  `https://libceed.readthedocs.io`
  and Jupyter-notebook tutorials
- BDDC preconditioners
- 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`

University of Colorado Boulder

Outlook

Ongoing and future work:

- Our (very first!) user manual `https://libceed.readthedocs.io` and Jupyter-notebook tutorials
- BDDC preconditioners
- 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!

University of Colorado Boulder

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:

$$\mathbf{B} = B \otimes B \otimes B \tag{11}$$

$$\mathbf{D}_0 = D \otimes B \otimes B \tag{12}$$

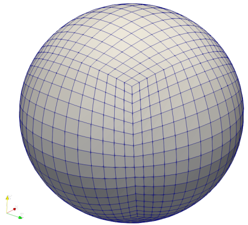
$$\mathbf{D}_1 = B \otimes D \otimes B \tag{13}$$

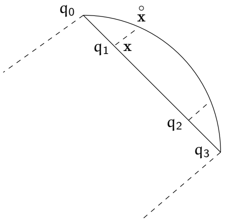
$$\mathbf{D}_2 = B \otimes B \otimes D \tag{14}$$

$$\mathbf{W} = W \otimes W \otimes W \tag{15}$$

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 $O(p^4)$.

Geometry on the sphere





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

$$\frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{X}}_{(3\times 2)} = \frac{\partial \overset{\circ}{\mathbf{x}}}{\partial \mathbf{x}}_{(3\times 3)} \frac{\partial \mathbf{x}}{\partial \mathbf{X}}_{(3\times 2)}$$

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

University of Colorado Boulder