libCEED

Valeria Barra

with: Jed Brown, Jeremy Thompson, Yohann Dudouit, and many others...

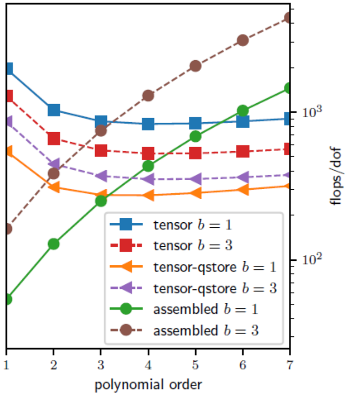
CU Boulder (CS)

For the CUCS-HPSC class
Oct. 25, 2019

CEED EXASCALE DISCRETIZATIONS

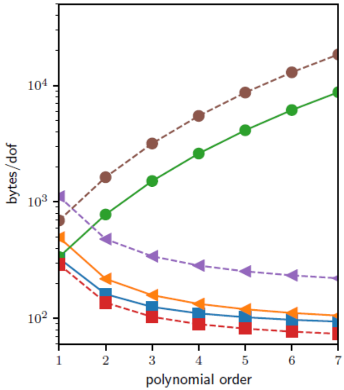
EXASCALE COMPUTING PROJECT

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



Flops per dof 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)

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



Memory bandwidth 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)

Overview

- 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 of accuracy 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 libCEED's performance and some examples

University of Colorado Boulder

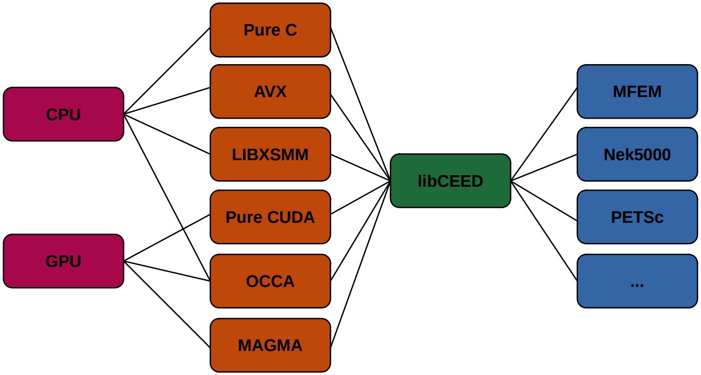
libCEED: the library of the CEED (Center for Efficient Extensible Discretizations)

- 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), v0.5 (September 2019)

For latest release:

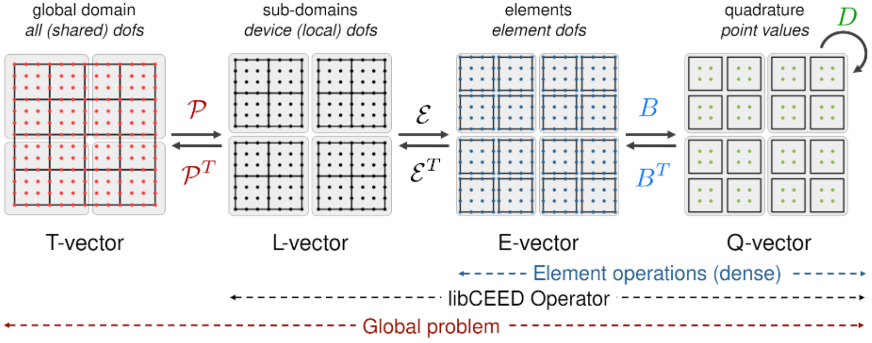
Tomov S., Abdelfattah A., Barra V., Beams N., Brown J. et al., *CEED ECP Milestone Report: Performance tuning of CEED software and 1st and 2nd wave apps* (2019, Oct 2$^{\mathrm{nd}}$) DOI: `https://doi.org/10.5281/zenodo.3477618`

libCEED backends



libCEED decomposition

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



The CEED project

CEED Bake-Off Problems

BP1: Solve {Bu=f}, where {B} is the mass matrix.

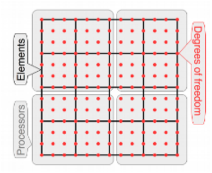
BP2: Solve the vector system {$Bu_i=f_i$} with {B} from BP1.

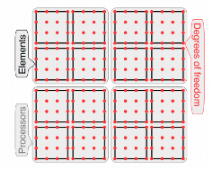
BP3: Solve {Au=f}, where {A} is the Poisson operator.

BP4: Solve the vector system {$Au_i=f_i$} with {A} from BP3.

- Range of polynomial orders: {p=1, 2,...,8}, at least.
- Cover range of sizes: from 1 element/MPI rank up to the memory limit.
- BP1 and BP2 are relevant for many hyperbolic substeps in transport problems. BP3 and BP4 reflect pressure, momentum, and diffusion updates in fluid/thermal transport.
- Vector forms BP2 and BP4 reveal benefits of increased *data reuse* and of *amortized communication overhead.*
- Benchmark repo: https://github.com/CEED/benchmarks

Exascale Computing Project





BP terminology: T- and E-vectors of HO dofs

EXASCALE COMPUTING PROJECT

University of Colorado Boulder

Composition of solvers for multiphysics problems

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

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

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

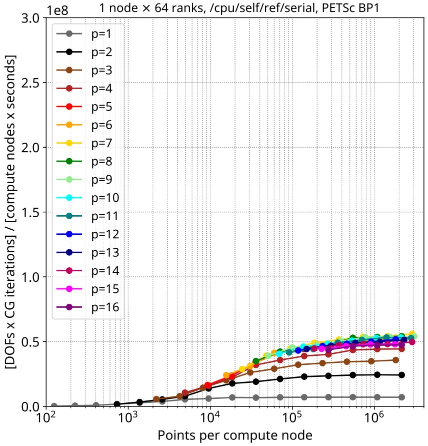
Ex: For a Dirichlet Stokes flow

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

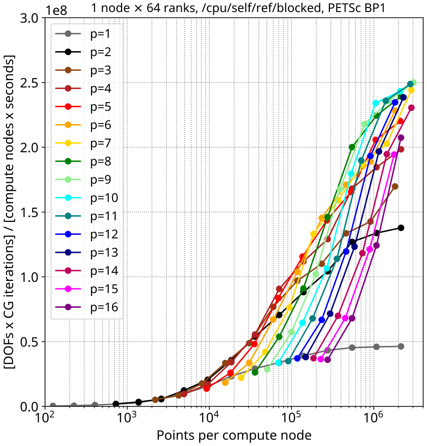
where $\boldsymbol{\sigma} = \mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T) - p\mathbf{I}_3$, and where the Schur's complement is $S = -J_{pu} J_{uu}^{-1} J_{pu}^T$ (needs preconditioning). If we use the simple block Jacobi preconditioner $\rightarrow$ block Gauss-Seidel, that can be solved by only partial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

University of Colorado Boulder

Let's talk performance



(a)

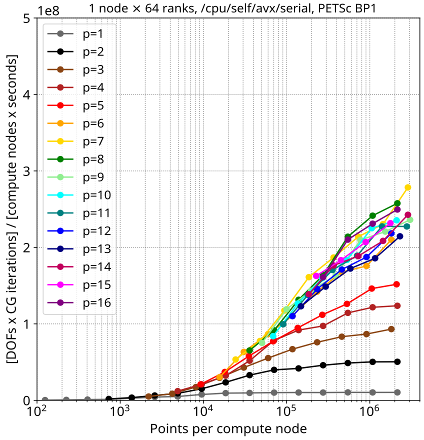


(b)

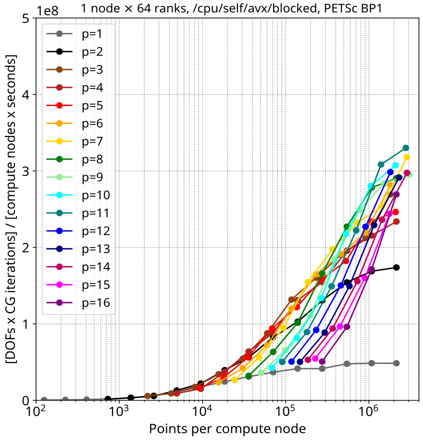
Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz). ref backend: In (a) serial implementation; in (b) blocked implementation.

University of Colorado Boulder

AVX backend



(a)

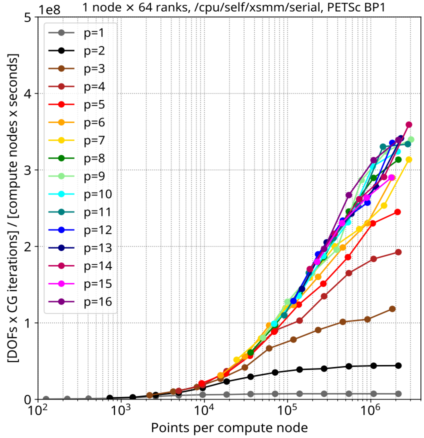


(b)

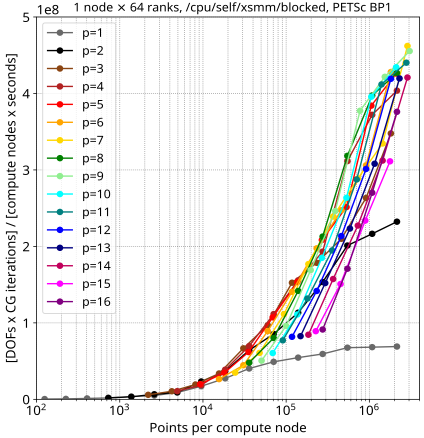
Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz). AVX backend: In (a) serial implementation; in (b) blocked implementation.

University of Colorado Boulder

libXSMM backend



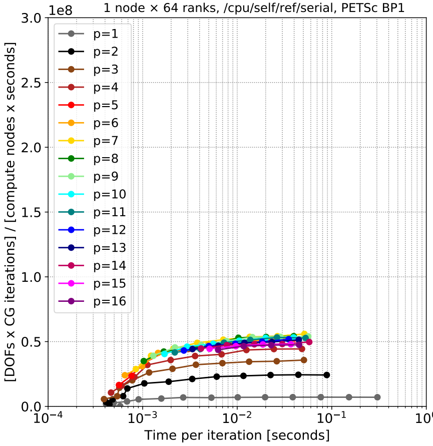
(a)



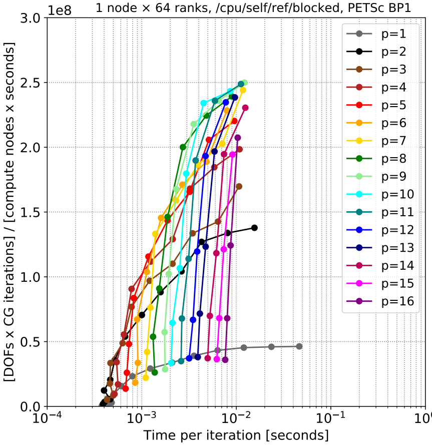
(b)

Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz). libXSMM backend: In (a) serial implementation; in (b) blocked implementation.

How fast?



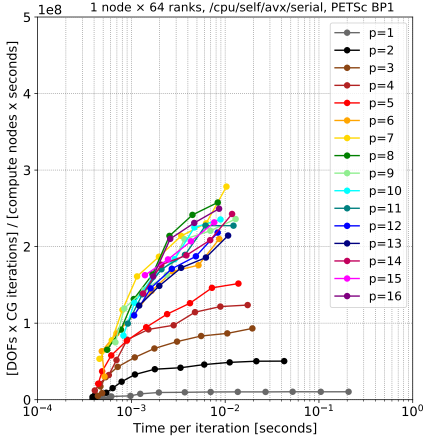
(a)



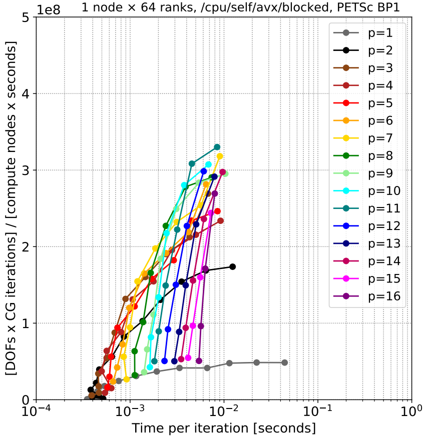
(b)

Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz). ref backend: In (a) serial implementation; in (b) blocked implementation.

AVX backend: How fast?



(a)

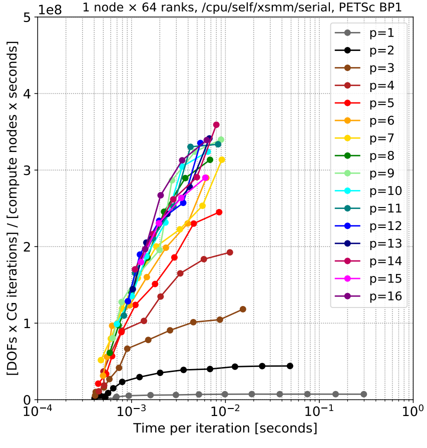


(b)

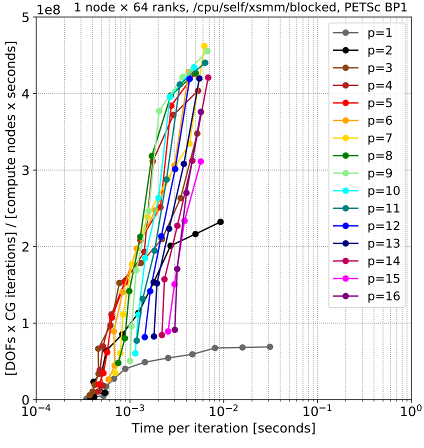
Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz). AVX backend: In (a) serial implementation; in (b) blocked implementation.

University of Colorado Boulder

libXSMM backend: How fast?



(a)

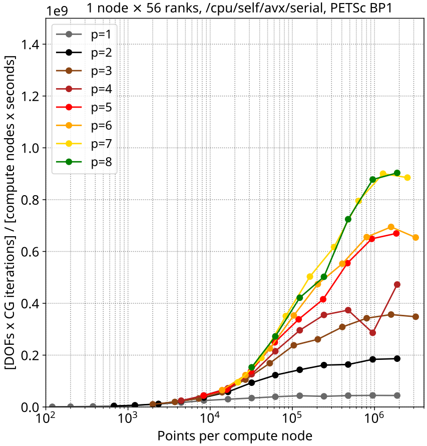


(b)

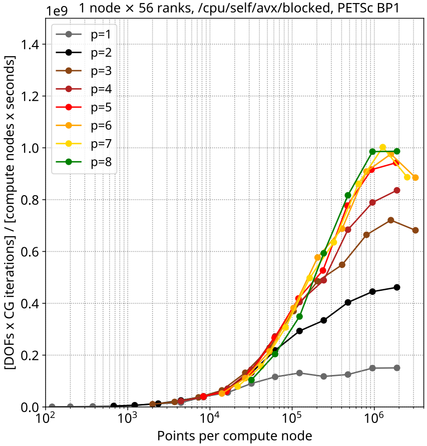
Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz). libXSMM backend: In (a) serial implementation; in (b) blocked implementation.

University of Colorado Boulder

Different architecture: Skylake. AVX backend



(a)

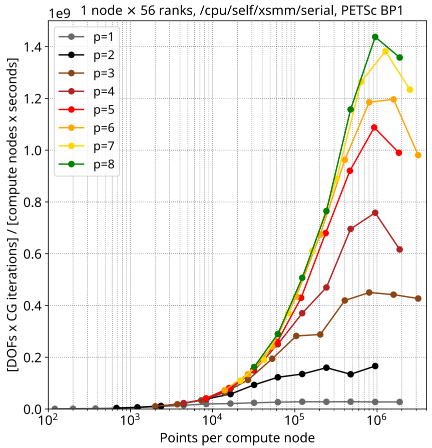


(b)

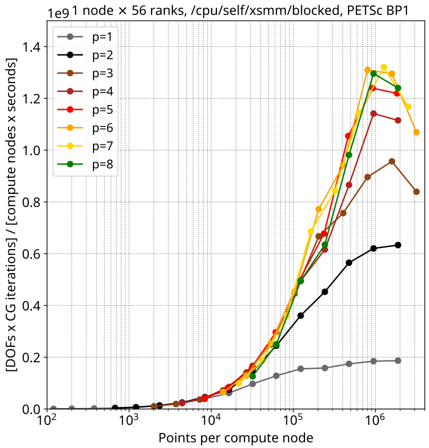
Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz). AVX backend: in (a) serial; in (b) blocked.

University of Colorado Boulder

Different architecture: Skylake. libXSMM backend



(a)

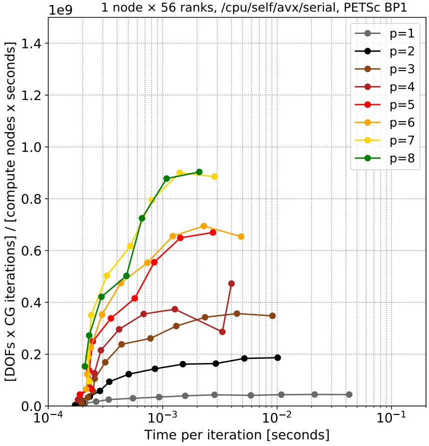


(b)

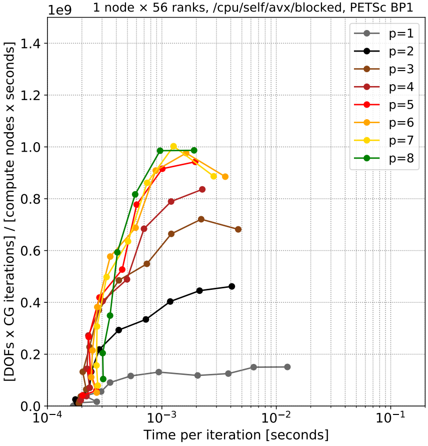
Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz). AVX backend: in (a) serial; in (b) blocked.

University of Colorado Boulder

How fast? Skylake: AVX backend



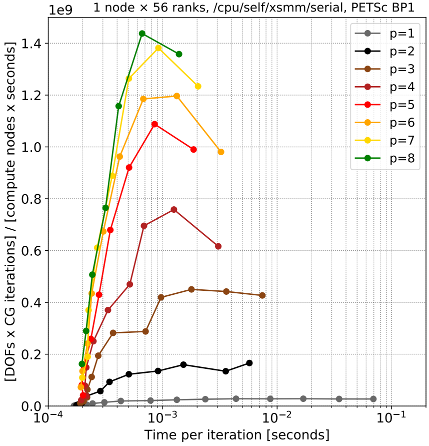
(a)



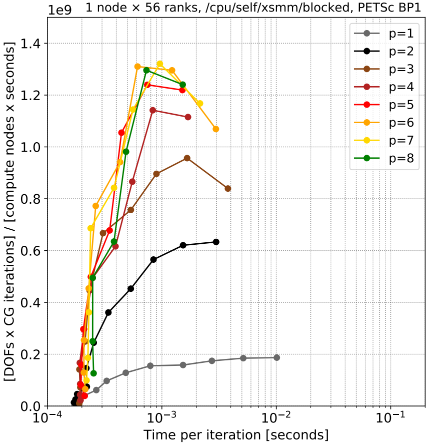
(b)

Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz). AVX backend: in (a) serial; in (b) blocked.

How fast? Skylake: libXSMM backend



(a)

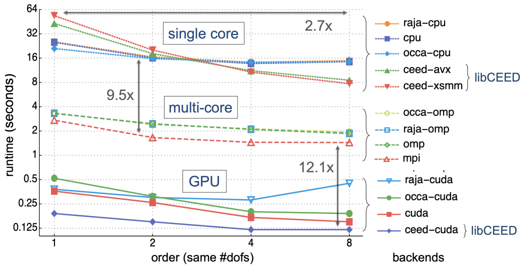


(b)

Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz). AVX backend: in (a) serial; in (b) blocked.

University of Colorado Boulder

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 =	**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**

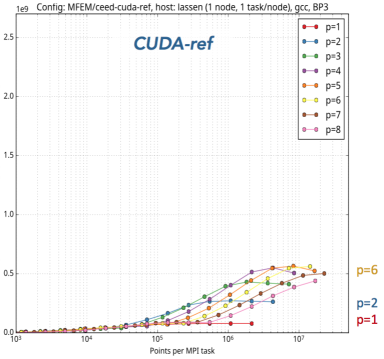
single-GPU, multi-core CPU, and single-core CPU,
for 1.3 millions DOFs in 2D.

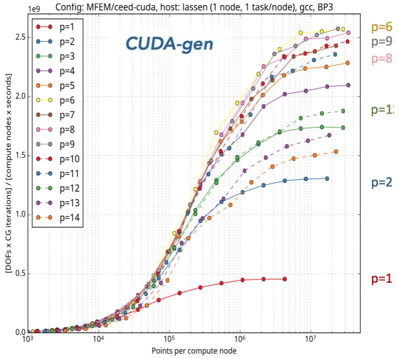
Legend: MFEM is a finite-element library; RAJA is a collection of C++ software abstractions; OCCA provides a unified API for different architectures and parallelism paradigms and uses just-in-time (JIT) compilation.

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.

University of Colorado Boulder

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





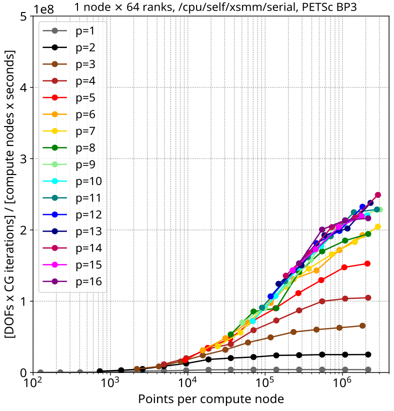
Legend: CUDA-ref is a libCEED backend that uses only simple reference pure CUDA kernels; CUDA-gen is an optimized libCEED backend that uses code generation (via JIT).

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

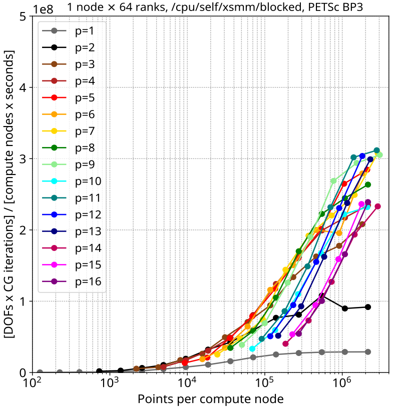
University of Colorado Boulder

Compare to libXSMM - our best CPU backend

On the KNL:



(a)

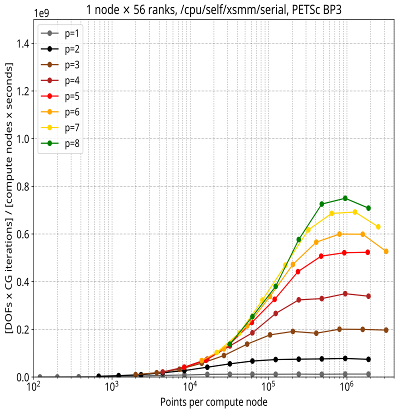


(b)

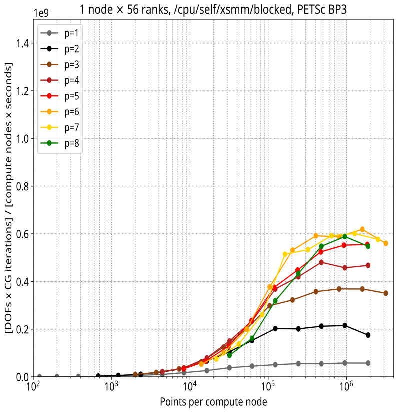
Figure: Knight Landing (Intel Xeon Phi 7230 SKU 1.3 GHz). AVX backend: in (a) serial; in (b) blocked.

Compare to libXSMM - our best CPU backend

And on the Skylake:



(a)



(b)

Figure: Skylake (2x Intel Xeon Platinum 8180M CPU 2.50GHz). AVX backend: in (a) serial; in (b) blocked.

University of Colorado Boulder

Towards a libCEED miniapp: a Navier-Stokes solver

Compressible Navier-Stokes equations in conservation form:

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

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

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

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

$(c_p/c_v - 1)\,(E - \mathbf{U} \cdot \mathbf{U}/(2\rho) - \rho g z) = P$	$\leftarrow$	pressure
μ	$\leftarrow$	dynamic viscosity
g	$\leftarrow$	gravitational acceleration
k	$\leftarrow$	thermal conductivity
λ	$\leftarrow$	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

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

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

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

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

Time stepping

For the Time Discretization we use an explicit formulation

$$\frac{\mathbf{q}_N^{n+1} - \mathbf{q}_N^n}{\Delta t} = -[\nabla \cdot \mathbf{F}(\mathbf{q}_N)]^n + [S(\mathbf{q}_N)]^n \,, \tag{4}$$

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

A very simple example: The advection equation

We analyze the transport of total energy

$$\frac{\partial E}{\partial t} + \nabla \cdot (\mathbf{u} E) = 0 \,, \tag{5}$$

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

$$\rho = \frac{P_0}{(c_p - c_v)\theta(\boldsymbol{x}, t)} \pi(\boldsymbol{x}, t)^{\frac{c_v}{c_p - c_v}} \,, \tag{6a}$$

$$e = c_v \theta(\boldsymbol{x}, t) \pi(\boldsymbol{x}, t) + \mathbf{u} \cdot \mathbf{u}/2 + gz \,, \tag{6b}$$

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

University of Colorado Boulder

Some work in progress

In collaboration with Prof. Jensen from CU Department of Aerospace Engineering Sciences (AES) we are working on Streamline upwind/Petrov-Galerkin (SUPG) stabilization of our Navier-Stokes example:

Not stabilizied version.

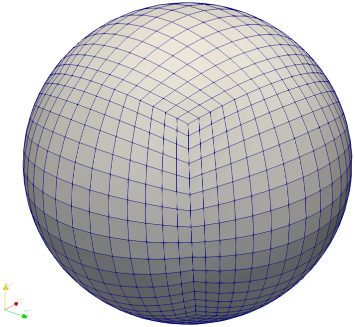
Stabilizied version.

Some other work in progress

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

$$\frac{\partial \mathbf{u}}{\partial t} = -(\omega + f)\hat{k} \times \mathbf{u} - \nabla \left(\frac{1}{2}|\mathbf{u}|^2 + g(h + h_s) \right) \tag{7a}$$

$$\frac{\partial h}{\partial t} = -\nabla \cdot (h_0 + h)\mathbf{u} \tag{7b}$$



Challenges:
Transform
$\mathbf{x} = (x, y, z) \in \mathbb{R}^3 \hookrightarrow$
$\widetilde{\mathbf{x}} = (\widetilde{x}, \widetilde{y}) \in \mathbb{R}^2 \hookrightarrow$
$\boldsymbol{\xi} = (\xi, \eta) \in \mathbf{I} = [-1, 1]^2$

University of Colorado Boulder

Thanks

Thank you!

University of Colorado Boulder