Performance and Portability with the libCEED Finite Element Library

Jeremy L Thompson

University of Colorado Boulder

jeremy.thompson@colorado.edu

September 18, 2018

Overview

A global sparse matrix is no longer a good representation of a high-order linear operator

libCEED is an extensible library that provides a portable algebraic interface and optimized implementations

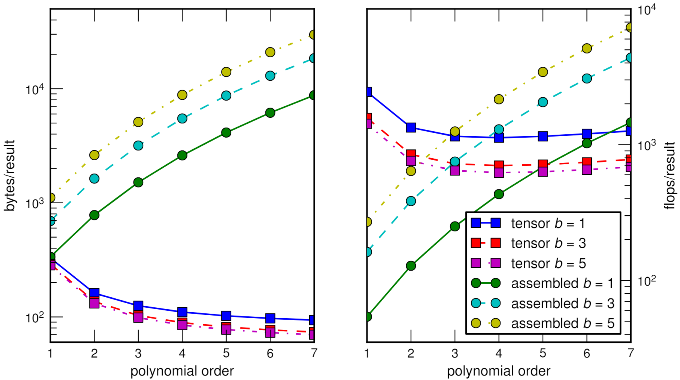
We have an example of portable and adaptable implementation with Navier-Stokes solver in libCEED and PETSc

We have results comparing performance on benchmark problems

Overview

1. Introduction
2. libCEED
3. Navier-Stokes
4. Benchmarks
5. Questions

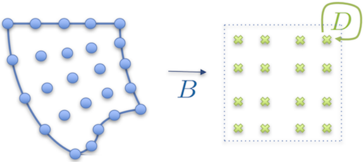
Assembled Matrix Cost



Memory bandwidth and ops per dof to apply a Jacobian from Q_k discretization of a b-variable PDE system using an assembled matrix versus matrix-free exploiting the tensor product structure

Matrix Free Implementation

$$\boxed{A = P^T G^T B^T D B G P}$$



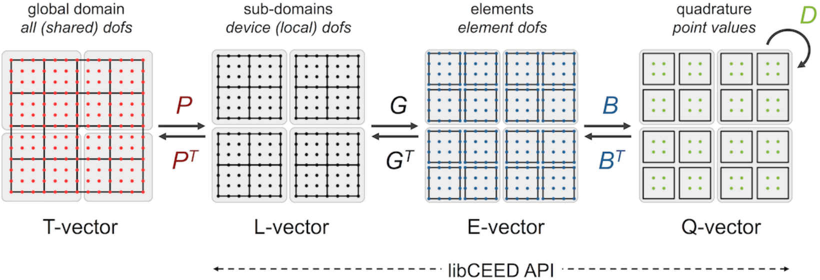
- Avoid global matrix assembly
- Map each element to reference element
- All data computed on the fly or precompute static data
- Easy to parallelize across nodes

libCEED API

- BSD-2 license, C99 library with F77 interface
- Releases: v0.2 March and v0.3 (imminent)
- Provides on-device operator implementation
- Easy to incorporate into existing code
- Supports multiple types of computational devices
  - CPU - Reference and blocked, template for new backends
  - OCCA (jit) - CPU, OpenMP, OpenCL, and CUDA
  - MAGMA
  - CUDA (in final development)
  - One source code can call multiple CEEDs with different backends

Operator Decomposition

$$A = P^T G^T B^T D B G P$$



API Objects

- G - CeedRestriction
  - Restrict to single element
  - User choice in ordering
- B - CeedBasis
  - Actions on basis such as interpolation, gradient, divergence, curl
  - Independent of geometry
- D - CeedQFunction
  - Operator action at quadrature points to include coefficient functions
  - Choice of when to compute metric terms and coefficents

Device Level Operator

- $L = G^T B^T D B G$ - CeedOperator
- libCEED objects are combined to create a CeedOperator
- CeedOperator gives operator action for elements on device
- User code responsible for communication between devices
  $A = P^T L P$

Quadrature Function

$$Au = F(v, u) = \int_\Omega v \cdot f_0(u, \nabla u) + \nabla v \cdot f_1(u, \nabla u)$$

$$Au = P^T G^T B^T D B G P u$$

- Quadrature function at the heart of the libCEED operator
- Multiple inputs and outputs
- Independent operations at quadrature points, ordering and number of elements not specified
- Code will be implementation agnostic

Navier-Stokes Formulation

- Compressible Navier-Stokes
- State variables: density, momentum, and total energy
- Boundary conditions:
  momentum - no-slip, non-penetrating
  density, energy - reflecting
- Initial conditions: Straka 1993
- Mesh: Box domain with hexehedral mesh

Strategy

- Implemented in PETSc and libCEED
- Setup phase computes geometric factors (Jacobian) and initial conditions
- Forward Euler for proof-of-concept version
- Compact: $\sim$ 480 lines of PETSc code and $\sim$ 200 lines of quadrature functions

Navier-Stokes QFunction

```
static int NS(void *ctx, CeedInt Q,
              const CeedScalar *const *in, CeedScalar *const *out) {
  // Inputs
  const CeedScalar *q = in[0], *dq = in[1], *qdata = in[2], *x = in[3];
  // Outputs
  CeedScalar *v = out[0], *vg = out[1];
...
  // Quadrature Point Loop
  for (CeedInt i=0; i<Q; i++) {
    // Setup
...
    // The Physics

    // -- Density
    // ---- u rho
    vg[i+(0+5*0)*Q] = rho*u[0]*BJ[0] + rho*u[1]*BJ[1] + rho*u[1]*BJ[2];
    vg[i+(0+5*1)*Q] = rho*u[0]*BJ[3] + rho*u[1]*BJ[4] + rho*u[1]*BJ[5];
    vg[i+(0+5*2)*Q] = rho*u[0]*BJ[6] + rho*u[1]*BJ[7] + rho*u[1]*BJ[8];

    // -- Momentum
...
    // -- Total Energy
    // ---- (E + P) u
    vg[i+(4+5*0)*Q]  = (E + P)*(u[0]*BJ[0] + u[1]*BJ[1] + u[2]*BJ[2]);
    vg[i+(4+5*1)*Q]  = (E + P)*(u[0]*BJ[3] + u[1]*BJ[4] + u[2]*BJ[5]);
    vg[i+(4+5*2)*Q]  = (E + P)*(u[0]*BJ[6] + u[1]*BJ[7] + u[2]*BJ[8]);
...
```

Bakeoff Problems

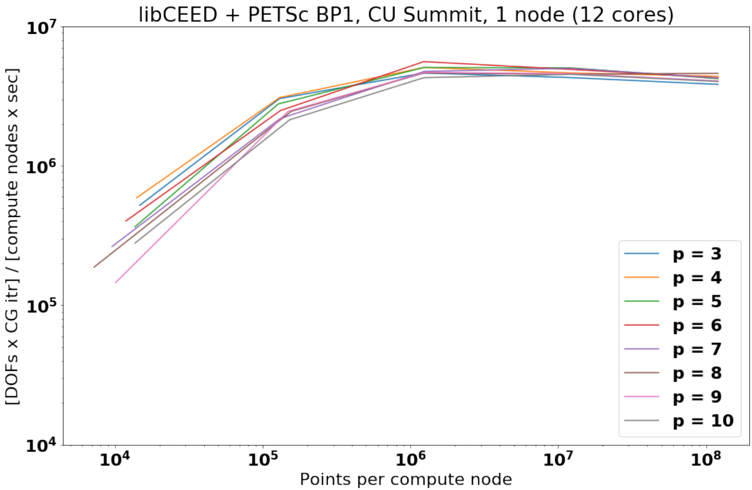
CEED Benchmark Problem 1
Problem: $\int vu = \int vf$ - L^2 projection

CEED Benchmark Problem 3
Problem: $\int v\Delta u = \int vf$ - Poisson

Domain: 3D Cube
Elements: Hexahedral
Shape Function Order: 2-10
Quadrature Points: 4^3-12^3
Machine: CU Boulder Summit

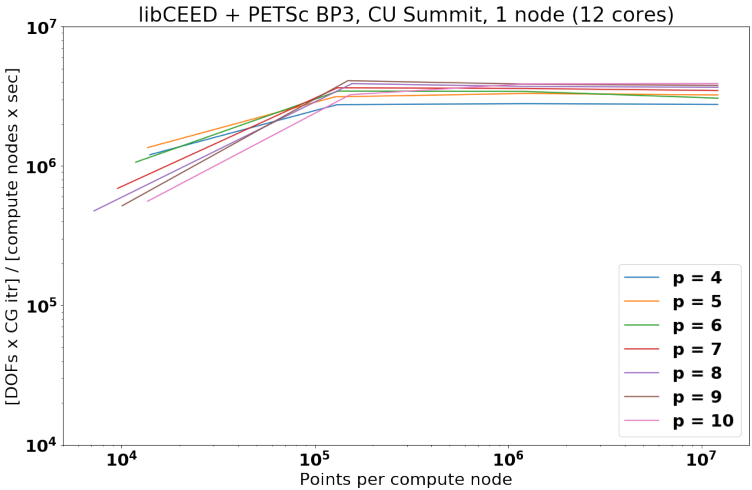
Nodes: 1
CPUs: Intel Xeon "Haswell"
Processors: 24 (12 used)
Compiler: Intel/17.4
MPI: Intel/17.3

BP1



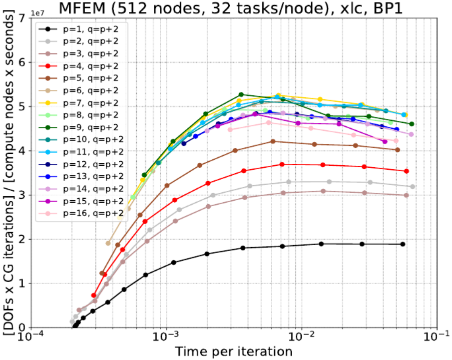
* Disclaimer - Results are very 'muddy'; Host code is not fully optimized and timing is for entire host code, with setup and destruction *

BP3

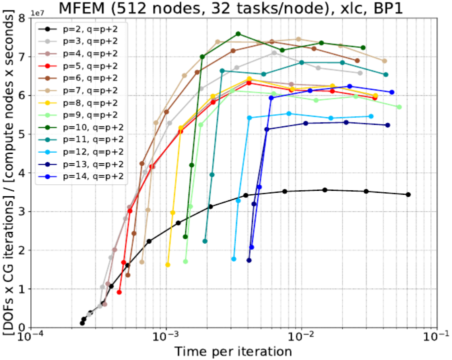


* Disclaimer - Results are very 'muddy'; Host code is not fully optimized and timing is for entire host code, with setup and destruction *

MFEM



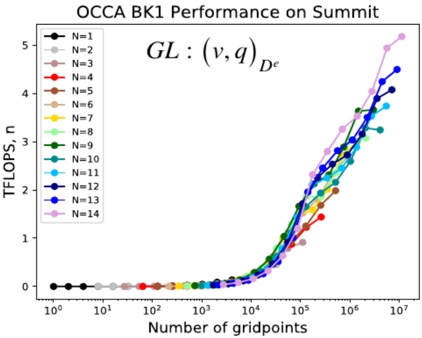
(a) Cetus internal vectorization

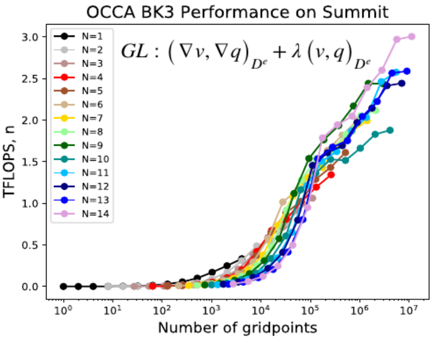


(b) Cetus external vectorization

Results by Thilina Rathnayake

OCCA on Summit





Results by Thilina Rathnayake

Future Work

- Continue performance tuning
- Improve GPU backends, reduce data movement
- Finalize pure CUDA backend
- Optimize additional geometries: tets, pyramids, and prisms
- Implement non-conforming meshes
- Create library of user quadrature functions
- Algorithmic differentiation of quadrature functions
- Composite operators, for mixed meshes and multiphysics
- Contributors and friendly users welcome

Questions?

Advisor: Jed Brown[1]

Collaborators: Valeria Barra [1], Jean-Sylvain Camier[2], Tzanio Kolev[2], Veselin Dobrev[2], Tim Warburton[3], David Medina[4], & Thilina Rathnayake[5]

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

1: University of Colorado, Boulder
2: Lawrence Livermore National Laboratory
3: Virginia Polytechnic Institute and State University
4: OCCA
5: University of Illinois, Urbana-Champaign

Performance and Portability with the libCEED Finite Element Library

Jeremy L Thompson

University of Colorado Boulder

jeremy.thompson@colorado.edu

September 18, 2018