On Performance and Portability for Generic Finite Element Interfaces

Jeremy L Thompson

University of Colorado Boulder

jeremy.thompson@colorado.edu

July 11, 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

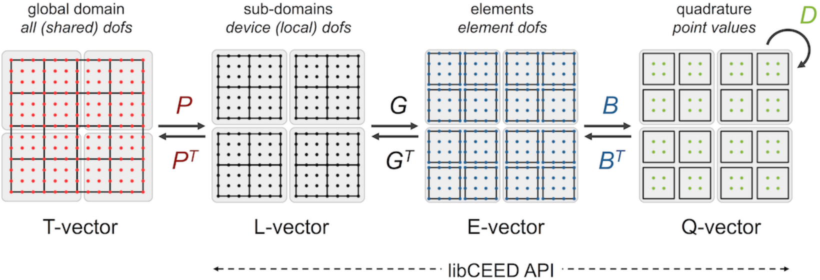
We have preliminary results comparing performance to native implementations in production software

Overview

1. Introduction
2. libCEED
3. Production Software
4. Performance Comparison
5. Questions

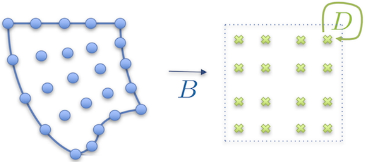
Operator Decomposition

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



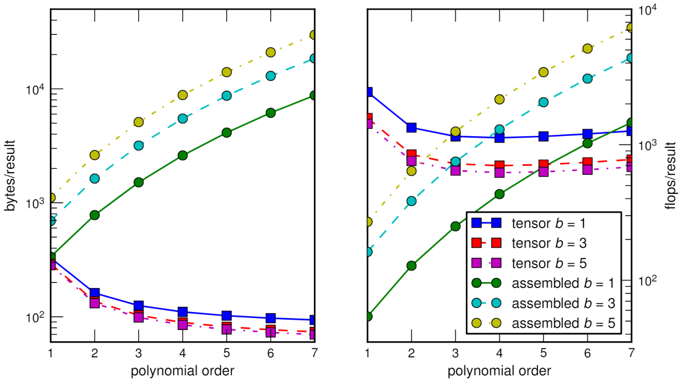
Matrix Free Implementation

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



- Avoid global matrix assembly
- Map each element to reference element
- Only store map to reference, action on reference
- Easy to parallelize across nodes

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

libCEED API

- Provides on-device operator implementation
- Easy to incorporate into existing code
- Supports multiple types of computatinal devices
  - CPU - Reference and vectorized, template for new backends
  - OCCA (jit) - CPU, OpenMP, OpenCL, and CUDA
  - MAGMA
  - One source code can call multiple CEEDs with different backends
- v0.2 March and v0.3 (imminent)
- BSD-2 license

API Objects

- G - CeedRestriction
  Restrict to single element
- B - CeedBasis
  Actions on basis such as interpolation, gradient, divergence, curl
- D - CeedQFunction
  Operator action at quadrature points to include coefficient functions

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

  $A = P^T L P$

Basis

- Tensor H^1 elements
- User provides p, q, dim and chooses Gauss or Gauss-Lobatto dofs
- Alternatively, user provides $1D$ interp, grad matrices and quadrature weights and points
- More geometries, $H(div)$, $H(curl)$ coming

```
CeedBasisCreateTensorH1Lagrange(ceed, dim, ncomp,
                                P, Q, CEED_GAUSS, &basis);
CeedBaisApply(basis, CEED_NOTRANSPOSE, CEED_EVAL_INTERP,
                                x, xq);
```

Restriction

- Gather and scatter operations
- Support conforming and non-conforming meshes
- User provides index list, may be linear combination of dofs
- On node communication only

```
CeedElemRestrictionCreate(ceed, ne, 2, ne+1, 1,
                          CEED_MEM_HOST, CEED_USE_POINTER,
                          ind, &rstr);
  CeedElemRestrictionApply(rstr, CEED_NOTRANSPOSE,
                          CEED_NOTRANSPOSE, u, ru,
                          CEED_REQUEST_IMMEDIATE);
```

Qfunction

- Applies the physics at the quadrature points
- Multiple inputs and outputs

```
CeedQFunctionCreateInterior(ceed, 1, myfunc,
                   __FILE__ ":myfunc", &qf);
CeedQFunctionAddInput(qf_setup, "_weight",
                   1, CEED_EVAL_WEIGHT);
CeedQFunctionAddInput(qf_setup, "x",
                   1, CEED_EVAL_GRAD);
CeedQFunctionAddOutput(qf_setup, "rho",
                   1, CEED_EVAL_NONE);
```

Operator

- Combines components to give local operator action
- Multiple inputs and outputs
- Composite operators coming

```
CeedOperatorCreate(ceed, qf_setup, NULL, NULL, &op_setup);

CeedOperatorSetField(op_setup, "_weight",
                     CEED_RESTRICTION_IDENTITY, basis,
                     CEED_VECTOR_NONE);
CeedOperatorSetField(op_setup, "x",
                     rstr, basis, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_setup, "rho",
                     CEED_RESTRICTION_IDENTITY,
                     CEED_BASIS_COLOCATED,
                     CEED_VECTOR_ACTIVE);
```

Benefits

- Extensible library
- Lower memory transfer, no sparse matrix
- Implementations for multiple devices and backends
- Backend improvements benefit all applications

    Tensor contraction, basis application, etc
- Minimal dependencies

Standalone Implementation

```
// Create the mass operator.
CeedOperator oper;
CeedOperatorCreate(CEED, apply_qfunc,
                   NULL, NULL, &oper);
```

...

```
// Apply the mass operator: 'u' -> 'v'.
CeedOperatorApply(oper, u, v,
                  CEED_REQUEST_IMMEDIATE);
```

MFEM

```
/// Wrapper for a mass CeedOperator as an
/// mfem::Operator
class CeedMassOperator : public mfem::Operator
 protected:
  const mfem::FiniteElementSpace *fes;
  CeedOperator build_oper, oper;
  CeedBasis basis, mesh_basis;
  CeedElemRestriction restr, mesh_restr;
  CeedQFunction apply_qfunc, build_qfunc;
  CeedVector node_coords, qdata;
```

Nek5000

```
subroutine ceed_axhm1(pap,ap1,p1,h1,h2,ceed,op_mass,
$ vec_ap1,vec_p1,vec_qdata)

include 'ceedf.h'

c Vector conjugate gradient matvec for solution of
c uncoupled Helmholtz equations

include 'SIZE'
include 'TOTAL'
...
call ceedvectorsetarray(vec_p1,ceed_mem_host,
$ ceed_use_pointer, p1,err)
call ceedoperatorapply(op_mass,vec_p1,vec_ap1,
$ ceed_request_immediate,err)
call ceedvectorgetarray(vec_ap1,ceed_mem_host,ap1,err)
```

PETSc

```
user->op = op_mass;
user->qdata = qdata;

ierr = MatCreateShell(comm, mdof[0]*mdof[1]*mdof[2],
       mdof[0]*mdof[1]*mdof[2],
       PETSC_DECIDE, PETSC_DECIDE, user, &mat);
CHKERRQ(ierr);
ierr = MatShellSetOperation(mat, MATOP_MULT
       (void(*)(void))MatMult_Mass); CHKERRQ(ierr);

...

ierr = KSPSetFromOptions(ksp); CHKERRQ(ierr);
ierr = KSPSetOperators(ksp, mat, mat); CHKERRQ(ierr);
ierr = KSPSolve(ksp, rhs, X); CHKERRQ(ierr);
```

Nek5000

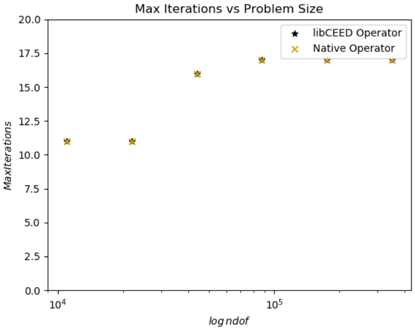
Problem: $\nabla u = f$
CEED Benchmark Problem 1

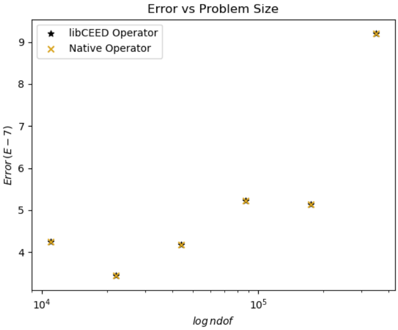
Domain: 3D Cube
Elements: Hexagonal
Number of Elements: 2^n
Shape Function Order: 7
Quadrature Points: 9

Computer: CU Boulder Summit

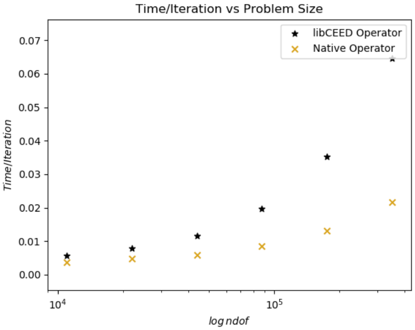
Nodes: 1
CPUs: Intel Xeon "Haswell"
Processors: 32
Compiler: Intel/17.0.0
MPI: Intel/2017.0.098

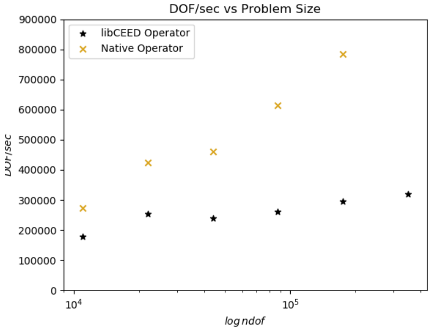
Nek5000 - The Good News





Nek5000 - The Bad News





Future Work

- Improve optimized CPU backend, vectorize across elements
- Improve GPU backends, reduce data movement
- Add additional geometries, tets, pyramids, and prisms
- Create library of user quadrature functions
- Composite operators, for mixed meshes and multiphysics
- Create pure CUDA backend
- Compare libCEED operators to native implementation in a wider range of production software
- Contributors and friendly users welcome

Questions?

Advisor: Jed Brown[1]

Collaborators: Jean-Sylvain Camier[2], Tzanio Kolev[2],
Veselin Dobrev[2], & Thilina Rathnayake[3]

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

1: University of Colorado, Boulder
2: Lawrence Livermore National Laboratory
3: University of Illinois, Urbana-Champaign

On Performance and Portability for Generic Finite Element Interfaces

Jeremy L Thompson

University of Colorado Boulder

jeremy.thompson@colorado.edu

July 11, 2018