Designing Generic Finite Element Interfaces

Jeremy L Thompson

University of Colorado Boulder

jeremy.thompson@colorado.edu

August 4, 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

Overview

Fitite Elements Operator

Finite elements discretizes the weak form of a PDE

Classical Form:
$-\Delta u = f$

Test Functions:

Weak Form:

Galerkin Form:

Fitite Elements Operator

Finite elements discretizes the weak form of a PDE

Classical Form:

$$-\Delta u = f$$

Test Functions:

$$-\int \varphi \Delta u = \int \varphi f$$

Weak Form:

Galerkin Form:

Fitite Elements Operator

Finite elements discretizes the weak form of a PDE

Classical Form:
$$-\Delta u = f$$

Test Functions:
$$-\int \varphi \Delta u = \int \varphi f$$

Weak Form:
$$\int \nabla \varphi \nabla u = \int \varphi f$$

Galerkin Form:

Fitite Elements Operator

Finite elements discretizes the weak form of a PDE

Classical Form:

$$-\Delta u = f$$

Test Functions:

$$-\int \varphi \Delta u = \int \varphi f$$

Weak Form:

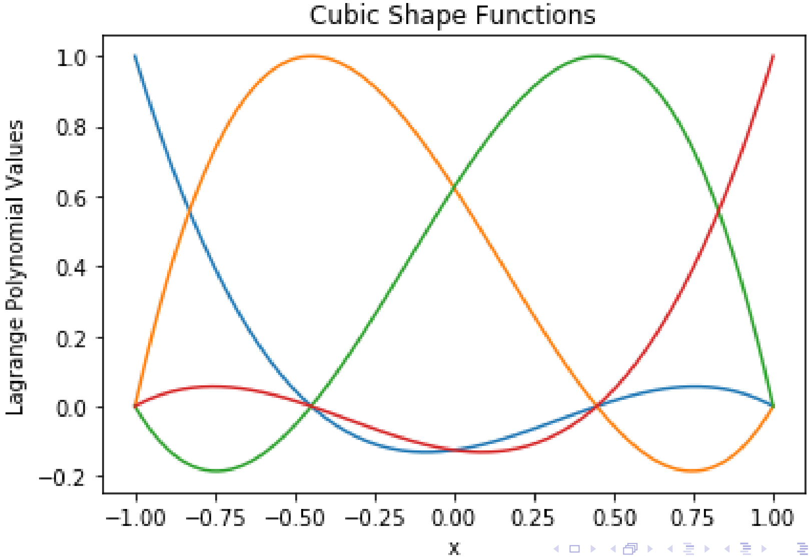
$$\int \nabla \varphi \nabla u = \int \varphi f$$

Galerkin Form:

$$\int \nabla \varphi_i \nabla u = \int \varphi_i f$$

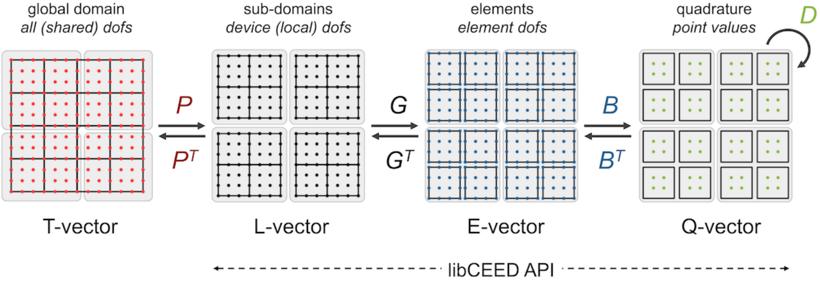
$$u = \sum_i c_i \varphi_i$$

Test Functions

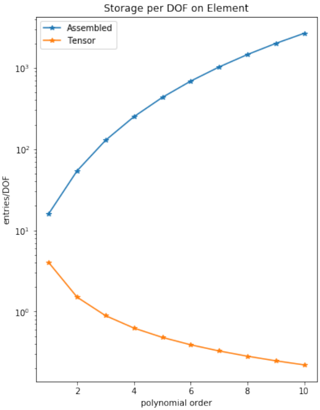


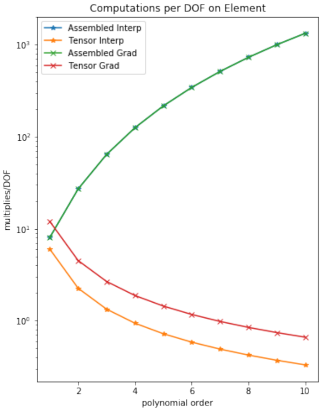
Operator Decomposition

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



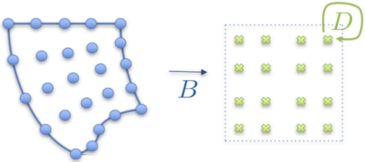
Assembled Matrix Cost!





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

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);
```

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

Designing Generic Finite Element Interfaces

Jeremy L Thompson

University of Colorado Boulder

jeremy.thompson@colorado.edu

August 4, 2018