Matrix Free Multigrid with libCEED
Challenges and Applications

Jeremy L Thompson

University of Colorado Boulder

jeremy.thompson@colorado.edu

Sept 27, 2019

libCEED Team

Developers: Jed Brown[1], Jeremy Thompson[1]
Thilina Rathnayake[2], Jean-Sylvain Camier[3], Tzanio Kolev[3], Veselin Dobrev[3], Valeria Barra[1], Yohann Doudouit[3], David Medina[4], Tim Warburton[5], & Oana Marin[6]

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

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

Overview

libCEED is an extensible library that provides a portable algebraic interface and optimized implementations of high-order operators

libCEED finite element operator decomposition provides opportunities for optimization and smart preconditioning

P-multigrid example offers insights into more flexible preconditioning techniques

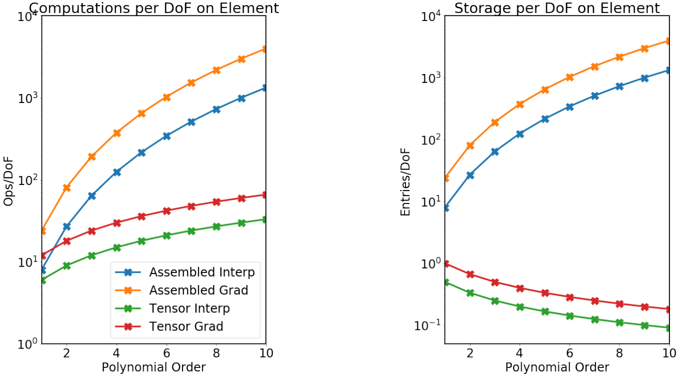
Overview

Center for Efficient Exascale Discretizations

DoE exascale co-design center

- Design discretization algorithms for exascale hardware that deliver significant performance gain over low order methods
- Collaborate with hardware vendors and software projects for exascale hardware and software stack
- Provide efficient and user-friendly unstructured PDE discretization component for exascale software ecosystem

Tensor Product Elements



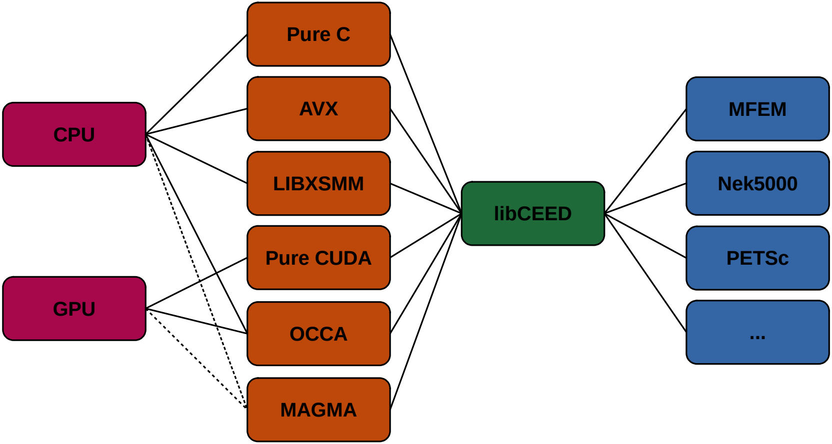
Using an assembled matrix forgoes performance optimizations for hexahedral elements

libCEED Design

libCEED design approach:

- Avoid global matrix assembly
- Optimize basis operations for all architectures
- Single source user quadrature point functions
- Easy to parallelize across hetrogeneous nodes

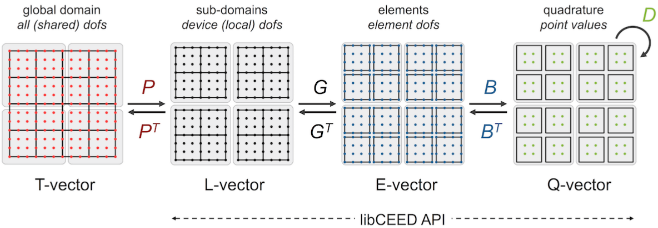
libCEED Backends



libCEED provides multiple backend implementations

libCEED Operator Decomposition

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



$$A_L = G^T B^T D B G$$

- G - CeedElemRestriction, local gather/scatter
- B - CeedBasis, provides basis operations such as interp and grad
- D - CeedQFunction, representation of PDE at quadrature points
- A_L - CeedOperator, aggregation of Ceed objects for local action of operator

Laplacian Example

Solving the 2D Poisson problem: $-\Delta u = f$
Weak Form: $\int \nabla v \nabla u = \int v f$

- General libCEED Operator
  $A_L = G^T B^T D B G$

- Laplacian Operator
  $A_L = G^T B^T_{Grad2D} D B_{Grad2D} G$

where D is block diagonal by quadrature point:

$$D_i = \left(w_i \det J_{geo} \right) J_{geo}^{-1} J_{geo}^{-T} \text{ and } J_{geo} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} \end{bmatrix}$$

x, y physical coords; r, s reference coords

Basis Optimization

Solving the 2D Poisson problem: $-\Delta u = f$
Weak Form: $\int \nabla v \nabla u = \int vf$

- General libCEED Operator
  $A_L = G^T B^T D B G$

- Laplacian Operator
  $A_L = G^T B_{Grad2D}^T D B_{Grad2D} G$

- Computationally Efficient Form
  $A_L = G^T \begin{bmatrix} B_G^T \otimes B_I^T & B_I^T \otimes B_G^T \end{bmatrix} D \begin{bmatrix} B_G \otimes B_I \\ B_I \otimes B_G \end{bmatrix} G$
  B_I - 1D Interpolation
  B_G - 1D Gradient

Basis Optimization

Solving the 2D Poisson problem: $-\Delta u = f$
Weak Form: $\int \nabla v \nabla u = \int vf$

- General libCEED Operator
  $A_L = G^T B^T D B G$

- Laplacian Operator
  $A_L = G^T B_{Grad2D}^T D B_{Grad2D} G$

- Computationally Efficient Form
  $A_L =$
  $$G^T (B_I^T \otimes B_I^T) \begin{bmatrix} \hat{B}_G^T \otimes I_2 & I_2 \otimes \hat{B}_G^T \end{bmatrix} D \begin{bmatrix} \hat{B}_G \otimes I_2 \\ I_2 \otimes \hat{B}_G \end{bmatrix} (B_I \otimes B_I) G$$
  where $\hat{B}_G = B_G B_I$

Operator Definition

General libCEED Operator:

$$v_L = A_L u_L$$

$$A_L = \mathbf{G}^\mathbf{T} B^T D B \mathbf{G}$$

Laplacian Operator Code:

```
CeedOperatorCreate(ceed, qf_apply, NULL, NULL, &op_apply);
CeedOperatorSetField(op_apply, "du", erestrictu, CEED_TRANSPOSE,
                     basisu, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op_apply, "geo",erestrictqdi,CEED_NOTRANSPOSE,
                     CEED_BASIS_COLLOCATED, geo);
CeedOperatorSetField(op_apply, "dv", erestrictu, CEED_TRANSPOSE,
                     basisu, CEED_VECTOR_ACTIVE);
...
CeedOperatorApply(op_apply, uloc, vloc, CEED_REQUEST_IMMEDIATE);
```

QFunction Definition

General libCEED QFunction:

$$v_q = D u_q$$

2D Laplacian QFunction:

$$\begin{bmatrix} dv_0 \\ dv_1 \end{bmatrix} = \begin{bmatrix} D_{00} & D_{01} \\ D_{01} & D_{11} \end{bmatrix} \begin{bmatrix} du_0 \\ du_1 \end{bmatrix}$$

2D Laplacian QFunction Code:

```
CeedQFunctionCreateInterior(ceed, 1, Poisson2D,
                            Poisson2D_loc, &qf_apply);
CeedQFunctionAddInput(qf_apply, "du", 2, CEED_EVAL_GRAD);
CeedQFunctionAddInput(qf_apply, "geo", 3, CEED_EVAL_NONE);
CeedQFunctionAddOutput(qf_apply, "dv", 2, CEED_EVAL_GRAD);
```

QFunction Definition

- Single Source QFunctions for all backends:
- C/C++ code, compiled with main for CPU, JiT for GPU

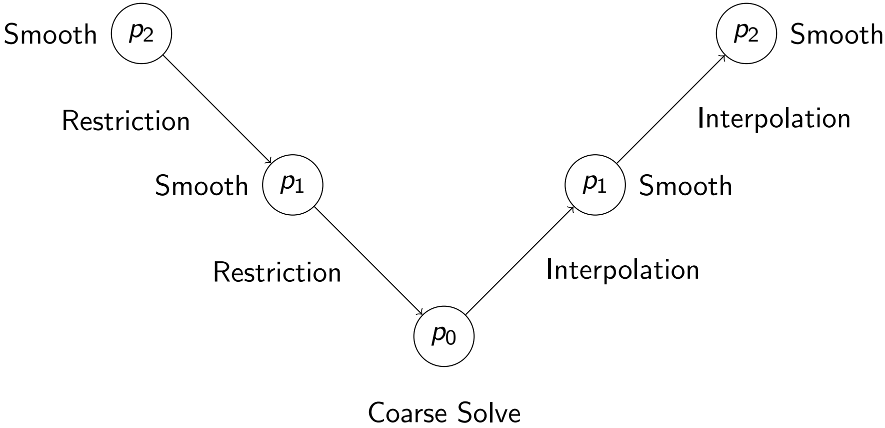
```
int Poisson2D(void *ctx, const CeedInt Q,
    const CeedScalar *const *in, CeedScalar *const *out) {
  // Inputs and Outputs
  const CeedScalar *du = in[0];
  CeedScalar *geo = out[0], *dv = out[1];

  // Quadrature Point Loop
  CeedPragmaSIMD // For CPU vectorization
  for (CeedInt i=0; i<Q; i++) {
    dv[i+Q*0] = geo[i+Q*0]*du[i+Q*0] + geo[i+Q*2]*du[i+Q*1];
    dv[i+Q*1] = geo[i+Q*2]*du[i+Q*0] + geo[i+Q*1]*du[i+Q*1];
  } // End of Quadrature Point Loop

  return 0;
}
```

P-Multigrid

- Preconditioning essential for iterative solvers, especially with high order
- Multigrid preconditioning offers $O(N)$ elliptic PDE solve
- H-multigrid difficult on unstructured/mixed meshes

V-Cycle

3 level multigrid example



libCEED Operators - Laplacian

Solving the 2D Poisson problem: $-\Delta u = f$
Weak Form: $\int \nabla v \nabla u = \int vf$

- General libCEED Operator
  $A_L = G^T B^T D B G$
- Laplacian Operator
  $A_L = G^T B_{Grad2D}^T D B_{Grad2D} G$
- Computationally Efficient Form
  $A_L =$
  $$G^T \left(B_I^T \otimes B_I^T \right) \begin{bmatrix} \hat{B}_G^T \otimes I_2 & I_2 \otimes \hat{B}_G^T \end{bmatrix} D \begin{bmatrix} \hat{B}_G \otimes I_2 \\ I_2 \otimes \hat{B}_G \end{bmatrix} \left(B_I \otimes B_I \right) G$$
  where $\hat{B}_G = B_G B_I$

libCEED Operators - Restriction

Restriction / Interpolation is largely a basis operation

- General libCEED Operator
  $A_L = G^T B^T D B G$
- Restriction / Interpolation Operator
  $A_L = G_c^T I I B_{ftoc} G_f$
- Computationaly Efficient Form
  $A_L = G_c^T \left(\hat{J}_{ftoc} \otimes \hat{J}_{ftoc} \right) G_f$

Performance

- 3D Poisson Problem
- Small test on personal laptop
- Mesh
  - 8^3 GLL points per element
  - Quadrature on 9^3 GL points per element
  - Cube with 32^3 elements, $89,200$ DoFs
- Unpreconditioned
  - 9.916e-06 $||\cdot||_\infty$ Error
  - 306 CG iterations
  - 11.838 million CG DoFs/sec
  - 2.306 sec CG solve time
- P-Multigrid
  - 9.916e-06 $||\cdot||_\infty$ Error
  - 59 CG iterations
  - 0.3211 million CG DoFs/sec
  - 16.390 sec CG solve time

Challenges

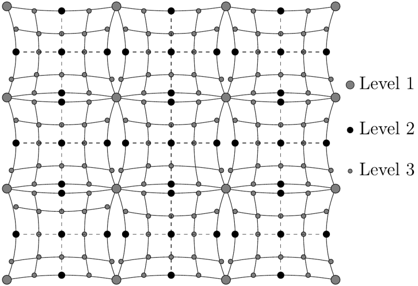
- Significantly decreased number of iterations... but
  - Iterations much slower than desired
  - Can decrease iteration time with lighter smoother
  - High order requires more levels, increased communication
- Caveats:
  - Small mesh run for demo purposes
  - PETSc code currently has issues running with CUDA in parallel

Way Forward

- Balancing Domain Decomposition by Constraints
- Designed for partially subassembled finite element operators
- Two levels, primal constraints and full mesh
- Lower communication, but heavier smoother required

Inexact Subdomain Solves

- Inexact subdomain solves (Li and Widlund 2007)
- Fast Diagonalization for inverses of separable operators
  - Used in some Additive Schwartz methods (Nek5000)
- Fast Diagonalization Method inspired subdomain approximate inverses



Future Work

- Further performance enhancements (GPU and CPU)
- Improved mixed mesh and operator composition support
- Expanded non-linear and multi-physics examples
- Preconditioning based on libCEED operator decomposition
- Algorithmic differentiation of user quadrature functions
- We invite contributors and friendly users

Questions?

Advisors : Jed Brown[1] & Daniel Appelö[1]

Collaborators: Valeria Barra[1], Oana Marin[2], Tzanio Kolev[3], Jean-Sylvain Camier[3], Veselin Dobrev[3], Yohann Doudouit[3], Tim Warburton[4], David Medina[5], & Thilina Rathnayake[6]

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

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

Matrix Free Multigrid with libCEED
Challenges and Applications

Jeremy L Thompson

University of Colorado Boulder

jeremy.thompson@colorado.edu

Sept 27, 2019