libCEED Finite Element Library
Development Update and Examples

Jeremy L Thompson
Valeria Barra, Jed Brown

University of Colorado Boulder

jeremy.thompson@colorado.edu

Sept 25, 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

We have optimized implementations for CPU and GPU

We have new performance optimizations, development in our example suite, and research in preconditioning strategies

Overview

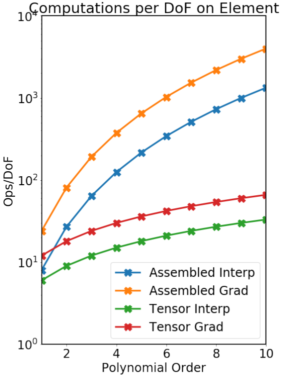
1. Introduction
2. libCEED
3. Example Suite
4. Current Efforts
5. Future Work
6. Questions

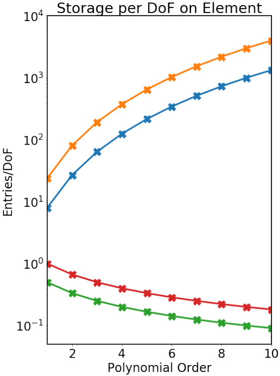
Center for Efficient Exascale Discretizations

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





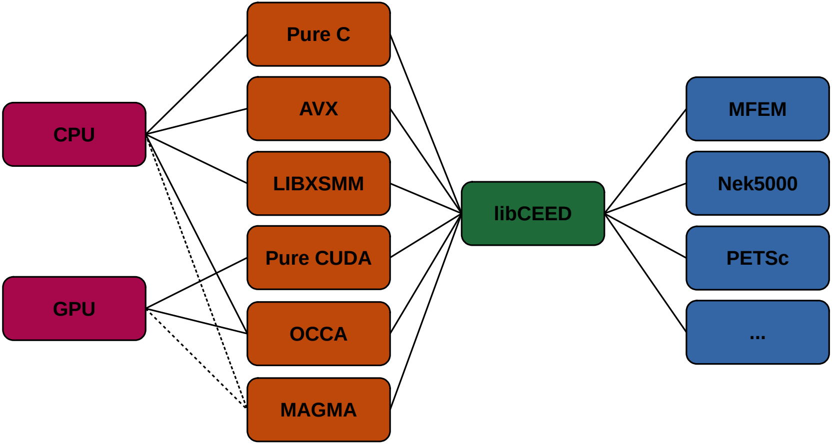
Using an assembled matrix forgoes performance optimizations for hexahedral elements

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

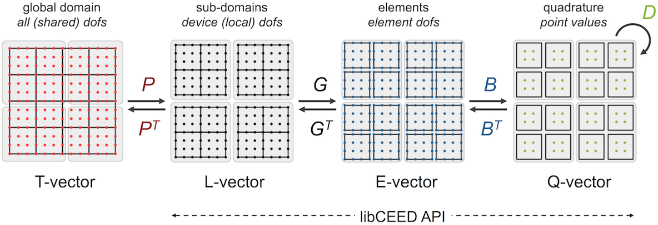
libCEED Backends



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: $\int \nabla v \nabla u = \int vf$

- 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
  $A_L =$
  $$G^T \left(B_I^T \otimes B_I^T \right) \left[\begin{array}{cc} \hat{B}_G^T \otimes I_2 & I_2 \otimes \hat{B}_G^T \end{array} \right] D \left[\begin{array}{c} \hat{B}_G \otimes I_2 \\ I_2 \otimes \hat{B}_G \end{array} \right] \left(B_I \otimes B_I \right) G$$
  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;
}
```

libCEED Performance

Benchmark performance across multiple implementations

Benchmark Problem 1/2:

- $Mu = f$
- L^2 projection problem

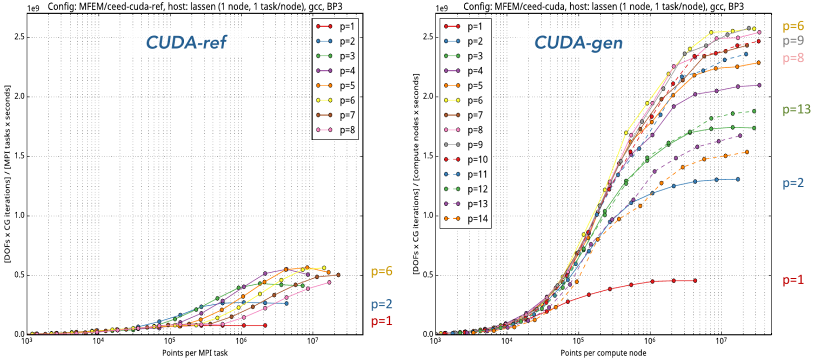
Benchmark Problem 3/4:

- $Ku = f$
- Poisson problem

3D scalar problem (BP 1/3) or 3D vector problem (BP 2/4)

Unpreconditioned CG, maximum of 20 iterations

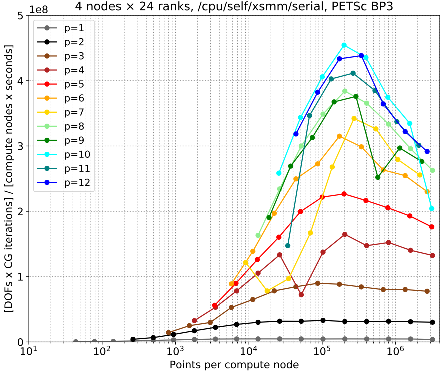
GPU Performance

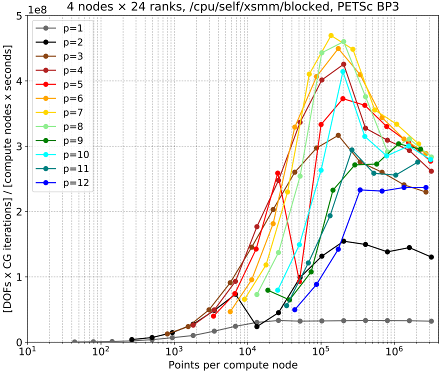


MFEM-BP3, 3D, Lassen 4 x V100 GPUs / node

- Substantial performance increase with Single Source QF + JiT
- +/- 10% performance of tuned kernels in libParanumal

CPU Performance





RMACC Summit, 4 × Intel Xeon E5-2680 v3

- External vectorization important at lower order
- Order we see performance 'switch' problem dependent

Navier-Stokes Example

- State Variables:
  - ρ - Mass density
  - U - Momentum density
  - E - Total Energy density

- 3D Compressible Navier-Stokes:

$$\frac{\partial \rho}{\partial t} + div\left(U\right) = 0$$
$$\frac{\partial U}{\partial t} + div\left(\rho\left(u \times u\right) + PI_3\right) + \rho g \hat{k} = div\left(F_u\right)$$
$$\frac{\partial E}{\partial t} + div\left(\left(E + P\right) u\right) = div\left(F_e\right)$$

- Viscous and Thermal Stresses:

$$F_u = \mu\left(\nabla u + \left(\nabla u\right)^T + \lambda div\left(u\right) I_3\right)$$
$$F_e = uF_u + k\nabla T$$

QFunction Assembly

User QFunction:

```
// ---- Fuvisc
const CeedInt Fuviscidx[3][3] = {{0, 1, 2}, {1, 3, 4}, {2, 4, 5}};
for (CeedInt j=0; j<3; j++)
  for (CeedInt k=0; k<3; k++)
    dv[k][j+1][i] -= wJ*(Fu[Fuviscidx[j][0]]*dXdxdXdxT[k][0] +
                         Fu[Fuviscidx[j][1]]*dXdxdXdxT[k][1] +
                         Fu[Fuviscidx[j][2]]*dXdxdXdxT[k][2]);
```

Assembly:

```
    dv[k][j+1][i] -= wJ*(Fu[Fuviscidx[j][0]]*dXdxdXdxT[k][0] +
b08d:   c5 7d 28 d0             vmovapd %ymm0,%ymm10
                         Fu[Fuviscidx[j][1]]*dXdxdXdxT[k][1] +
b091:   c4 42 c5 b8 d3          vfmadd231pd %ymm11,%ymm7,%ymm10
b096:   c5 fd 28 84 24 c8 04    vmovapd 0x4c8(%rsp),%ymm0
b09d:   00 00
    dv[k][j+1][i] -= wJ*(Fu[Fuviscidx[j][0]]*dXdxdXdxT[k][0] +
b09f:   c4 62 f5 ac 14 07       vfnmadd213pd (%rdi,%rax,1),%ymm1,%ymm10
b0a5:   c5 7d 11 14 07          vmovupd %ymm10,(%rdi,%rax,1)
                         Fu[Fuviscidx[j][1]]*dXdxdXdxT[k][1] +
b0aa:   c5 7d 59 94 24 68 04    vmulpd 0x468(%rsp),%ymm0,%ymm10
b0b1:   00 00
...
```

Example Suite

Ongoing development in example suite

- PHASTA investigating porting to libCEED
    - SUPG stabilization
    - Primitive variable formulation
    - Implicit time integrator
- Initial development of shallow water equations example

Preconditioning

Iterative solvers require preconditioning

Especially with high-order finite element operators

- Operator Diagonal
  - Diagonally dominant operators
- P-Multigrid
  - Elliptic operators
- BDDC with FDM
  - In development

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

libCEED Finite Element Library Development Update and Examples

Jeremy L Thompson
Valeria Barra, Jed Brown

University of Colorado Boulder

jeremy.thompson@colorado.edu

Sept 25, 2019