libCEED: A Case Study in Hidden Benefits of the Bridge Pattern

Jeremy L Thompson,
Natalie Beams, Jed Brown, and Yohann Dudouit

University of Colorado Boulder

jeremy.thompson@colorado.edu

July 31, 2020

libCEED Team

Developers: Ahmad Abdelfattah[1], Valeria Barra[2], Natalie Beams[1], Jed Brown[2], Jean-Sylvain Camier[3], Veselin Dobrev[3], Yohann Dudouit[3], Leila Ghaffari[2], Tzanio Kolev[3], David Medina[4], Thilina Rathnayake[5], Jeremy L. Thompson[2], & Stan Tomov[5]

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

1: University of Tennesse
2: University of Colorado, Boulder
3: Lawrence Livermore National Laboratory
4: Occalytics LLC
5: University of Illinois at Urbana-Champaign

Overview

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

We have optimized implementations for multiple architectures

Bridge design pattern offers performance portability
as well as improved debugability and internal design documentation

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

libCEED Philosophy

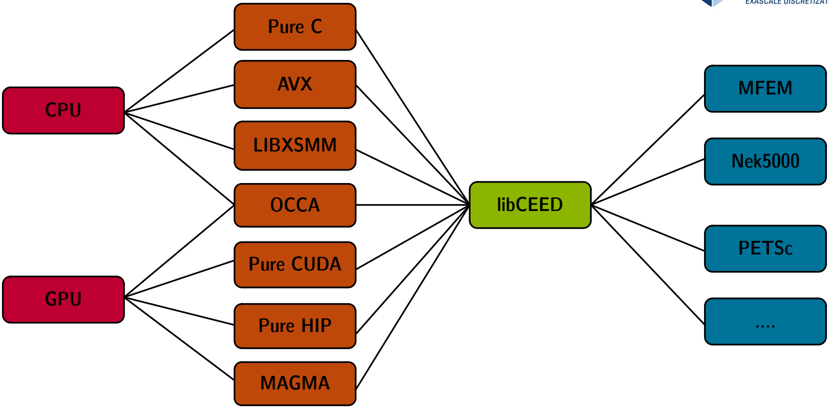
libCEED provides purely algebraic interface for matrix-free evaluation of arbitrary polynomial order PDE operators

libCEED design approach:

- Optimized implementations for multiple architectures
- Runtime selection of backend implementation
- Single source user PDE quadrature point functions

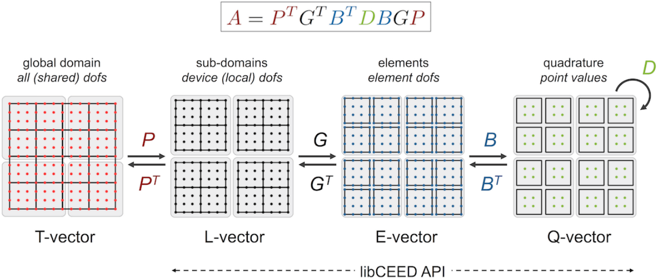
Repository: https://github.com/CEED/libCEED

libCEED Backends



Natural use case for bridge pattern

libCEED Operator Decomposition



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

- 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

User QFunction Source

User QFunction Code:

```c
// ---- 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]);
```

Compiled 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
```

Similar optimized code on other architectures

End Goal: Optimized Backends

CPU:

- AVX instructions for vector operations, FMA
- LIBXSMM for JiT optimized small matrix multiplication

GPU:

- JiT to generate device kernels from user code
- Fused kernels to minimize launches and data movement

Development Challenges

CPU:

- Optimized code can be difficult to read
- External library calls can be opaque

GPU:

- Debugging more challenging on devices
- Fused kernels can't be incrementally tested

Bridge Pattern FTW

Strong encapsulation of backend implementation details from interface

- Reference backends provide straightforward implementation
  - /cpu/self/ref/serial
  - /gpu/cuda/ref
  - /gpu/hip/ref
- Debugging backends provide additional debugging tools
  - /cpu/self/memcheck
- Optimized backends selectively re-implement objects
  - /cpu/self/xsmm/blocked
  - /gpu/cuda/gen

Hidden Benefits

- Stable API and ABI across implementations
- Flexible and distributed development
- Clear reference code for new developers
- Improved debugability ,for users and developers
- ...

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] & Adriana Gillman[1]

Developers: Ahmad Abdelfattah[1], Valeria Barra[2], Natalie Beams[1], Jed Brown[2], Jean-Sylvain Camier[3], Veselin Dobrev[3], Yohann Dudouit[3], Leila Ghaffari[2], Tzanio Kolev[3], David Medina[4], Thilina Rathnayake[5], & Stan Tomov[5]

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

1: University of Tennesse
2: University of Colorado, Boulder
3: Lawrence Livermore National Laboratory
4: Occalytics LLC
5: University of Illinois at Urbana-Champaign

libCEED: A Case Study in Hidden Benefits of the Bridge Pattern

Jeremy L Thompson,
Natalie Beams, Jed Brown, and Yohann Dudouit

University of Colorado Boulder

jeremy.thompson@colorado.edu

July 31, 2020