CEED Software Thrust and libCEED

Jed Brown, Jeremy Thompson, Valeria Barra (CU Boulder)
Yohann Dudouit, Jean-Sylvain Camier, Veselin Dobrev, and Tzanio Kolev (LLNL)
Misun Min, Oana Marin (ANL)
David Medina (Two Sigma/LLNL)
Thilina Rathnayake and Paul Fischer (University of Illinois)

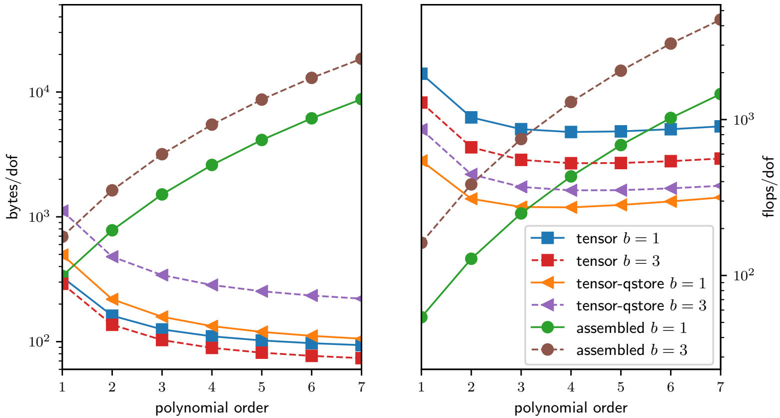
CEED Annual Meeting, 2019-08-06

CEED 2.0 Software release (March)

- GSLIB-1.0.2
- HPGMG-0.4
- Laghos-2.0
- libCEED-0.4
- MAGMA-2.5.0
- MFEM-3.4
- Nek5000-17.0
- Nekbone-17.0
- NekCEM-7332619
- PETSc-3.11.1
- PUMI-2.2.0
- OCCA-1.0.8

- `spack install ceed`
- `docker run -it -rm -v ‘pwd‘:/ceed jedbrown/ceed bash`
- `sbatch --image docker:jedbrown/ceed job.sh`
  In job.sh: `srun -n 64 shifter ./your-app`

Resource requirements for assembled versus unassembled

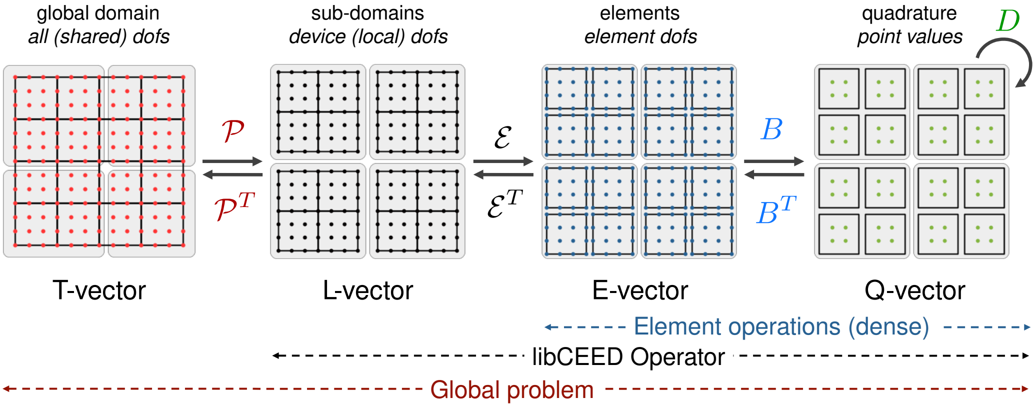


- Arithmetic intensity for Q_p elements
  - $< \frac{1}{4}$ (assembled), ≈ 10 (unassembled), ≈ 5 to 10 (hardware)
- store Jacobian information at Gauss quadrature points, can use AD

libCEED: Code for Efficient Extensible Discretization

- BSD-2 license, C library with Fortran interface
- Releases: v0.1–v0.3 (2018), v0.4 (March), v0.5 (imminent)
- Purely algebraic interface
- Extensible backends
  - CPU: reference, vectorized, XSMM
  - CUDA using NVRTC (Yohann Dudouit)
  - OCCA (just-in-time compilation): CPU, OpenMP, OpenCL, CUDA
  - MAGMA
- Platform for collaboration with vendors
- Minimal assumptions about execution environment, parallel decomposition
- Primary target: high order finite element methods
  - $H^1, H(\mathrm{div}), H(\mathrm{curl})$
  - also of interest to spectral difference, etc.
  - Exploit tensor product structure when possible

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



Quadrature Function

$$v^T F(u) \sim \int_\Omega v \cdot f_0(u, \nabla u) + \nabla v : f_1(u, \nabla u) \qquad v^T J w \sim \int_\Omega \begin{bmatrix} v \\ \nabla v \end{bmatrix}^T \begin{bmatrix} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{bmatrix} \begin{bmatrix} w \\ \nabla w \end{bmatrix}$$

$$u = B_I \mathscr{E}_e u_L \qquad \nabla u = \frac{\partial X}{\partial x} B_\nabla \mathscr{E}_e u_L$$

$$J w = \sum_e \mathscr{E}_e^T \begin{bmatrix} B_I \\ B_\nabla \end{bmatrix}^T \underbrace{\begin{bmatrix} I & \\ & \left(\frac{\partial X}{\partial x}\right)^T \end{bmatrix} W_q \begin{bmatrix} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{bmatrix} \begin{bmatrix} I & \\ & \left(\frac{\partial X}{\partial x}\right) \end{bmatrix}}_{\text{coefficients at quadrature points}} \begin{bmatrix} B_I \\ B_\nabla \end{bmatrix} \mathscr{E}_e w_L$$

- B and B_∇ are tensor contractions – independent of element geometry
- Choice of how to order and represent gathers $\mathscr{E}$ and scatters $\mathscr{E}^T$
- Who computes the metric terms and other coefficients?
- Similar for Neumann/Robin and nonlinear boundary conditions

Quadrature Functions

- Multiple inputs and outputs
- Independent operations at each of `Q` quadrature points
  - Ordering and number of elements not specified

```
int L2residual(void *ctx, CeedInt Q,
               const CeedScalar *const in[],
               CeedScalar *const out[]) {
  const CeedScalar *u = in[0], *rho = in[1], *target = in[2];
  CeedScalar *v = out[0];
  for (CeedInt i=0; i<Q; i++)
    v[i] = rho[i] * (u[i] - target[i]);
  return 0;
}
CeedQFunctionAddInput(qf, "u", 1, CEED_EVAL_INTERP);
CeedQFunctionAddInput(qf, "rho", 1, CEED_EVAL_INTERP);
CeedQFunctionAddInput(qf, "target", 1, CEED_EVAL_INTERP);
CeedQFunctionAddOutput(qf, "v", 1, CEED_EVAL_INTERP);
```

Building Operators

$$v^T F(u) \sim \int_\Omega v \cdot f_0(u, \nabla u) + \nabla v : f_1(u, \nabla u)$$

$$u = B_I \mathcal{E}_e u_L \quad \nabla u = \frac{\partial X}{\partial x} B_\nabla \mathcal{E}_e u_L$$

- `CeedOperatorCreate(ceed,` f`, &op);`
- `CeedOperatorSetField(op, "velocity",` $\mathcal{E}$`,` B`,` u_L`);`
- Apply: implementation handles batching, work buffers, and calling $f(u, \nabla u)$.
- User links to interface library
- Backend implementation switchable at run-time
- Two-phase implementation enables connectivity analysis and JIT

libCEED Operator

$$A = \mathscr{P}^T \underbrace{\mathscr{E}^T B^T D B \mathscr{E}}_{\text{CeedOperator}} \mathscr{P}$$

- quadrature function D
- For each field: element restriction $\mathscr{E}$, basis B, where to find vector

```
CeedOperatorCreate(ceed, qf_L2residual, &op);
CeedOperatorSetField(op, "u", E, Basis, CEED_VECTOR_ACTIVE);
CeedOperatorSetField(op, "rho", E_id, CEED_BASIS_COLOCATED, rho);
CeedOperatorSetField(op, "target", E_id, CEED_BASIS_COLOCATED, target)
CeedOperatorSetField(op, "v", E, Basis, CEED_VECTOR_ACTIVE);

CeedOperatorApply(op, u, v, &request);
```

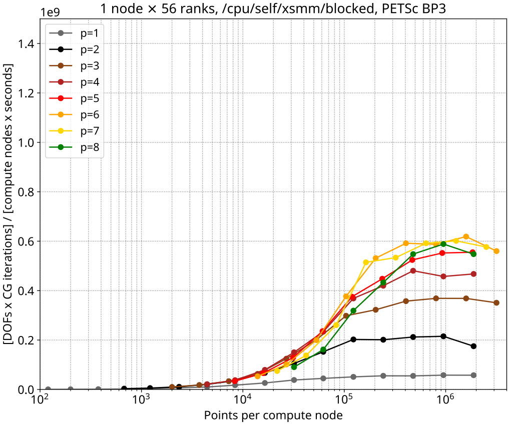
Element restriction $\mathcal{E}_e$

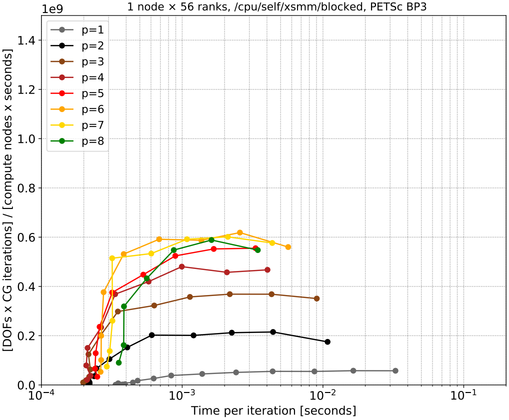
- Conforming homogeneous mesh: boolean matrix with homogeneous block size
- Non-conforming mesh: anchored rows have linear combination
- Nek5000/DG-style E-vector: indexed identity
- libCEED backends are allowed to reorder, compress, etc.
- May be applied all at once or in batches
- CeedOperator implementation chooses if/how to fuse

Vectorization techniques

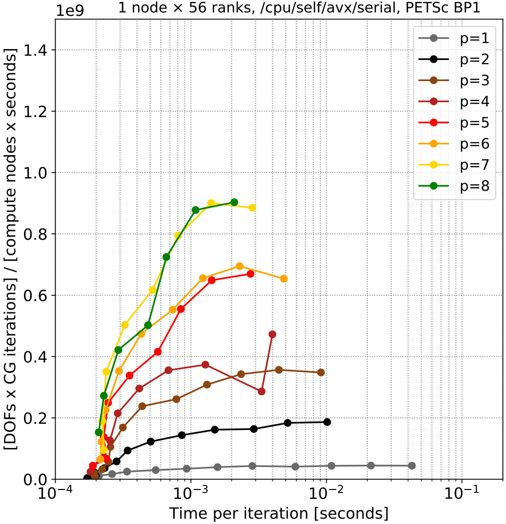
- Vectorize within a single high-order element
  - Minimal working set (as small as one element)
  - Specialized implementation for different degree/# quadrature points
  - Hard to avoid cross-lane operations at modest degree
  - Nek5000
- Vectorize across elements in batches `[i,j,k,e]`
  - Working set has at least vector length number of elements (e.g., 8)
  - Generic implementation is easy to optimize; no cross-lane operations
  - HPGMG-FE, Deal.II (Kronbichler and Kormann), MFEM (new)

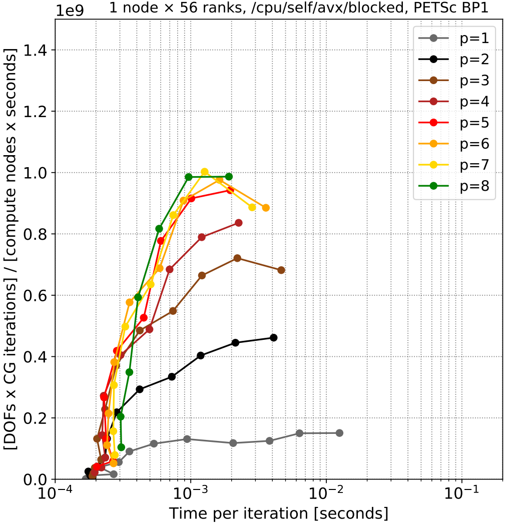
BP1: Skylake AVX *n* vs *t*



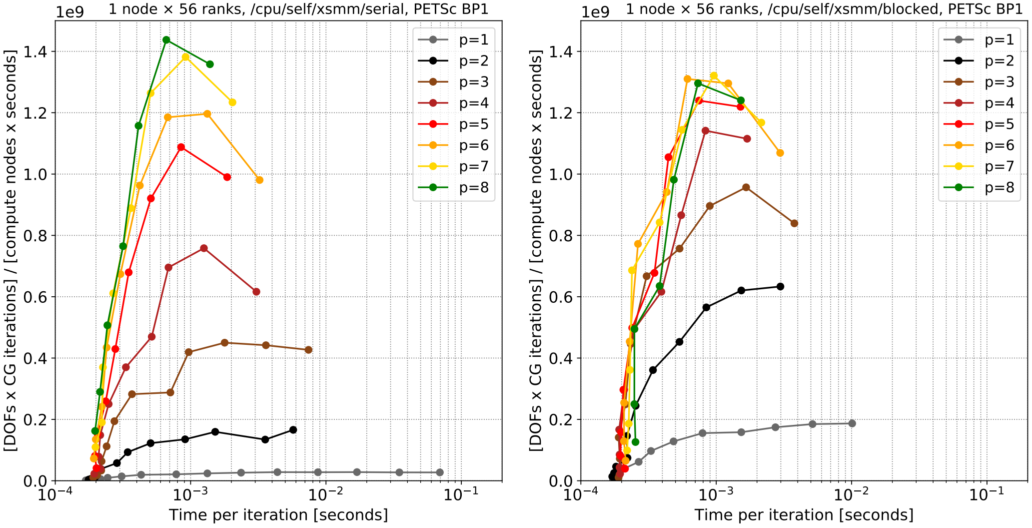


BP1: Skylake AVX

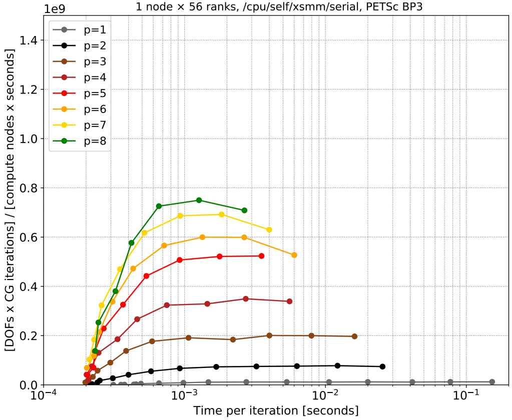




BP1: Skylake libXSMM



BP3: Skylake libXSMM



GPU support

In-place with CPU (host) memory

```
PetscScalar *y;
VecGetArray(Ypetsc, &y);
CeedVectorSetArray(Yceed, CEED_MEM_HOST, CEED_USE_POINTER, y);
CeedOperatorApply(op, Xceed, Yceed, CEED_REQUEST_IMMEDIATE);
```

In-place with GPU (device) memory

```
PetscScalar *y;
VecCUDAGetArray(Ypetsc, &y);
CeedVectorSetArray(Yceed, CEED_MEM_DEVICE, CEED_USE_POINTER, y);
CeedOperatorApply(op, Xceed, Yceed, CEED_REQUEST_IMMEDIATE);
```

- All PETSc Krylov methods run entirely on the device
- Preconditioning options limited compared to CPU
- (New) intra-node communication optimization and smarter packing (Junchao Zhang)

Operator composition

tests/t520-operator.c

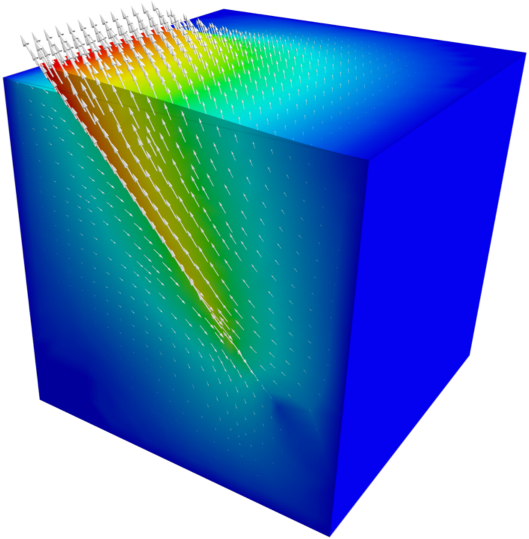
```
CeedCompositeOperatorCreate(ceed, &op_mass);
CeedCompositeOperatorAddSub(op_mass, op_massTet);
CeedCompositeOperatorAddSub(op_mass, op_massHex);

CeedOperatorApply(op_mass, U, V, CEED_REQUEST_IMMEDIATE);
```

- Backend can group within like topologies; flatten iteration space

Unified qfunctions: PR 304 (Yohann)

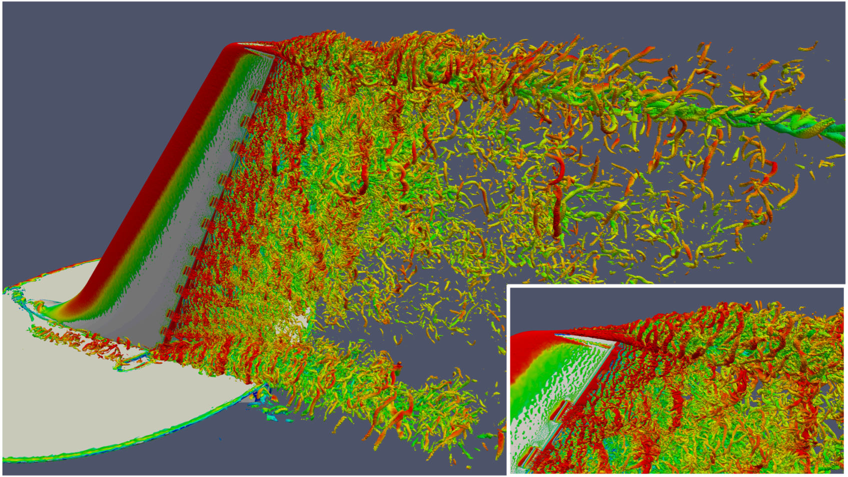
```
CEED_QFUNCTION int Diff(void *ctx, const CeedInt Q,
                        const CeedScalar *const *in,
                        CeedScalar *const *out) {
  const CeedScalar *ug = in[0], *qd = in[1];
  CeedScalar *vg = out[0];
  for (CeedInt i=0; i<Q ; ++i) {
    const CeedScalar ug0 = ug[i+Q*0];
    const CeedScalar ug1 = ug[i+Q*1];
    const CeedScalar ug2 = ug[i+Q*2];
    vg[i+Q*0] = qd[i+Q*0]*ug0 + qd[i+Q*1]*ug1 + qd[i+Q*2]*ug2;
    vg[i+Q*1] = qd[i+Q*1]*ug0 + qd[i+Q*3]*ug1 + qd[i+Q*4]*ug2;
    vg[i+Q*2] = qd[i+Q*2]*ug0 + qd[i+Q*4]*ug1 + qd[i+Q*5]*ug2;
  }
  return 0;
}
```

libCEED apps beyond ECP: PyLith



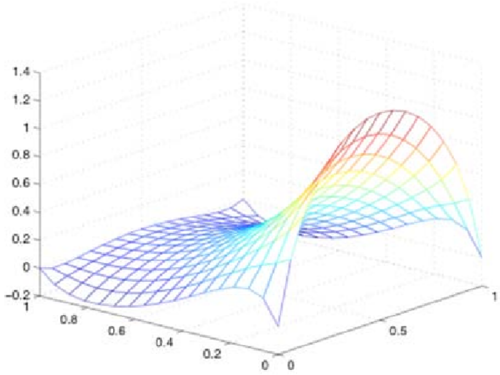
- Quasi-static and dynamic rupture simulations
- Rate and state fault models; extensible using Python, large user community
- Relatively low order currently; numerical dispersion for rupture
- Joe Geisz: CU summer undergrad integrating libCEED into PyLith

libCEED apps beyond ECP: PHASTA



- DDES for flow control, cardiovascular flow (SimVascular)
- Leila Ghaffari: CU starting PhD student; Valeria, Oana, Ken Jansen

Adaptive BDDC: robust coarsening/smoothing



[Mandel and Sousedík (2010)]

- weak continuity between subdomains
- orthogonality between "smoother" and coarse corrections
- adaptive coarse spaces; sharp convergence guarantees
- $\kappa \sim \left(1 + \log p^2\right)^2$ versus p^2 (ASM)

libCEED Outlook

- Parallel examples: MFEM, Nek5000, PETSc
- Ongoing optimization: CPU, CUDA (Tier 1); OCCA, MAGMA, OpenMP/OpenACC
  - Automatic choice of strategy
- Faster/more automatic mixed topology and mixed order meshes
- Gallery of Q-functions in development
- Algorithmic differentiation for Q-functions
- Preconditioning
  - FDM as smoothers, interface to PETSc (Jeremy, Oana)
  - Smoothers for non-separable operators
  - Adaptive BDDC using libCEED (with Stefano Zampini, KAUST)
- FMS and visualization